Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Document

D-99-01

"May I Speak Freely?" Between Templates and Free Choice in Natural Language Generation

Workshop at the 23rd German Annual Conference for Artificial Intelligence (KI '99), Bonn

14.–15. September 1999

Tilman Becker, Stephan Busemann (eds.)

August 1999

Deutsches Forschungszentrum für Künstliche Intelligenz

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: + 49 (631) 205-3211
Fax: + 49 (631) 205-3210
E-Mail: info@dfki.uni-kl.de

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: + 49 (681) 302-5252
Fax: + 49 (681) 302-5341
E-Mail: info@dfki.de

WWW: http://www.dfki.de

"May I Speak Freely?" Between templates and free choice in Natural Language Generation

Tilman Becker, Stephan Busemann (eds.)

DFKI-D-99-01

ISSN 0946-0098

“May I Speak Freely?”

Between Templates and Free Choice in Natural Language Generation

Workshop at the 23rd German Annual Conference for Artificial Intelligence (KI '99)
Bonn, 14.–15. September 1999

edited by

Tilman Becker
Stephan Busemann

DFKI GmbH

Saarbrücken, August 1999

Contents

Introduction

Natural language generation (NLG) technology is currently finding its way into commercial systems. Promising applications are the automatic generation of weather forecasts, information about measurement data, or various kinds of authoring systems that help an author in composing a text.
Generally the techniques used in NLG applications and application-oriented NLG systems differ from those utilized in research systems. While the latter typically aim at general, in-depth solutions, the former are geared towards solving particular classes of NLG problems. This involves shallow generation such as dealing with canned texts or templates rather than choosing freely from the coverage of complex linguistic grammars. Correspondingly, different systems encode linguistic knowledge at different levels of detail and sophistication.
In spite of recent, more unified theoretical accounts of the NLG process, these differences persist. This is, we believe, to a large extent due to different requirements posed by the application tasks.
While many theoretical linguistical concepts are difficult to use in practice, the techniques used in application systems often lack theoretical foundation. Their linguistic inadequacy severely restricts the transportability of systems to other tasks and domains. Current work in NLG addresses this problem, e.g., the RAGS project (`http://www.itri.brighton.ac.uk/projects/rags`). Whatever the chosen approach, its adequacy depends on the expected input to, and the desired output of, the NLG system. Input may vary from non-linguistic data to surface-semantic sentence representations. Further generation parameters may be available, such as a user model or a discourse model. Output may vary from a single sentence per context to multiple alternatives that allow, or require, the user to choose from.

This workshop is, to our knowledge, the first one topicalizing the relation between application tasks and technologies used. It aims at exploring the tension between more general and more specific approaches to NLG, thereby clarifying what NLG technology is suited best for which task. It is intended to be an opportunity to get an overview over existing state-of-the-art technology and its optimal usage. It will be relevant for both developers and users of NLG systems. Exploring conditions for successful NLG applications is a step that should be taken jointly by technology providers and current and potential users of NLG software. The invited speaker, Paul Heisterkamp of DaimlerChrysler AG, will focus on the industrial usage of NLG software, and we appreciate his contribution to this volume.
The workshop is embedded into the German Annual AI conference KI'99, following its tradition of hosting small hot-topic workshops. At the same time it is an activity of the Special Interest Group for Natural Language Systems (Fachgruppe 1.3.1) of the German association for computer science, Gesellschaft für Informatik (GI). Calling for international contributions was successful, as we have ten papers from six European countries and from the U.S. It was also mandatory since the topic in hand could not be covered adequately by national attendance only.
The contributions to this volume are unpublished research reports reviewed by the workshop organizers. The authors agreed to make available to each other the submitted papers before preparing the final versions. The papers can also be downloaded from the workshop's web page at `http://www.dfki.de/service/NLG/KI99.html`. The full report is available electronically from the DFKI library (`http://www.dfki.de/dfkibib/index.html`).

We now provide an overview of the paper contributions. The first four papers address the relation between shallow and in-depth generation. Van Deemter, Kramer and Theune (p. 1) restate it as plan-based versus template-based generation and attack the widespread assumption that plan-based approaches are theoretically well-founded whereas template-based generation is application-dependent and lacks a theoretical basis. Using the template-based system GoalGetter as an example, which comments on soccer games, it is shown how templates can encode linguistically relevant information.
Reiter (p. 7) sees shallow techniques as a necessity where we lack sufficient knowledge of deeper techniques or don't have the resources to create the expensive in-depth software for a particular task. With the STOP system, which produces smoking cessation leaflets, he demonstrates that shallow techniques can beneficially be employed for some constraint enforcement and optimisation tasks.
Like van Deemter *et al.*, Bateman and Henschel (p. 13) argue against the supposed dichotomy between in-depth and shallow approaches to NLG. They emphasize that there is a continuum between the two, giving rise to hybrid systems. This is supported by their method of automatically compiling customized subgrammars from a general, linguistically well-founded grammar. The compiled grammars may vary in complexity; they may correspond to templates.
Calder, Evans, Mellish, and Reape offer yet another perspective on the same idea (p. 19). They sketch a uniform, architectural framework called whiteboard, in which various kinds of representation structures, partial ones or mixed ones, co-exist. They have a derivational history based on the modules that were needed to generate them. The whiteboard may also contain canned structures (e.g., canned texts) which differ from others in that they have no derivational history.

Heisterkamp (p. 25) discusses requirements for the commercial use of NLG systems, concentrating on generation for spoken dialogue systems. Besides real-time processing, ellipsis generation, reformulation, choice of mode, and time-alignment for multi-modal systems, the importance of application building and maintenance tools is pointed out.

The next three papers present solutions for various applications or application scenarios. Buchin and Schmerl (p. 31) describe the system TextPro, which generates technical and mathematical texts in multiple languages from a formal language. Expressions of the formal language are computed from an disambiguated quasi-natural language that is more amenable to the author of a text.
Formal descriptions are also a topic in Dalianis' contribution (p. 36). Complicated formal descriptions of domains mainly within the manufacturing industry are difficult to understand and need to be paraphrased in terms of a NL. The paper discusses the Volvex system in which the user can ask for concept descriptions that are based on both planning and template techniques.
Fritsch, Cousin and Tanguy (p. 42) developed a template-based system designed to improve multi-lingual interactive abilities in the Internet needed for activities such as reservations or virtual shopping. In a given context, the user, after entering a keyword, selects from a set of generated sentences one that meets her goal. A corresponding sentence in the target language is then generated and transmitted.

The last three papers deal with NLG in spoken dialogues. Geldof (p. 48) describes context-sensitive language production by a wearable device, the "COMRIS parrot." She discusses

various mechanisms to produce variable, parameterized output within a template-based approach.
A speech act-based account of turn-taking in free-flowing task-oriented dialogues forms the basis of Stent's contribution (p. 52) in which she sketches the specific requirements on an NLG component. She suggests that dialogue grounding and turn-taking are candidates for template-based realization, whereas task-oriented and supporting acts should be based on planning.
Santamarta (p. 58) describes work on the generation component in the Swedish LinLin dialogue system framework. She argues for the use of a customisable domain-dependent planner and a general realiser. The use of templates is considered risky in a dialogue system.

In addition to the paper presentations, some systems will be demonstrated, among them STOP by Reiter and the COMRIS generation by Geldof.

We believe that the contributions in this volume provide a representative picture of the present discussions on the proper use of NLG technology, ranging from theoretical discussions to practical implementations. The outcome of such discussions will be influenced by both theoretical insights and industrial requirements, and it will provide feedback to both NLG technology and future applications. We feel that this topic should be taken up again at an international workshop on NLG, which we will organise at Dagstuhl Castle, Germany, in July 2000 (preceding COLING 2000).

Tilman Becker
Stephan Busemann

Workshop Program

Tuesday, 14 September 1999

15:30–15:45	**Tilman Becker, Stephan Busemann,** DFKI Saarbrücken: *Welcome and Introductory Remarks*
15:45–16:15	**Kees van Deemter**, ITRI, Brighton, **Emiel Kramer, Mariët Theune**, IPO, Eindhoven: *Plan-based vs. template-based NLG: a false opposition?*
16:15–16:45	**Ehud Reiter**, University of Aberdeen: *Shallow vs. Deep Techniques for Handling Linguistic Constraints and Optimisations*
16:45–17:15	**John Bateman,** University of Bremen, **Renate Henschel,** HCRC, University of Edinburgh: *From full generation to 'near-templates' without loosing generality*
17:15–17:45	**Jo Calder,** University of Edinburgh, **Roger Evans,** University of Brighton, **Chris Mellish, Mike Reape,** University of Edinburgh: *"Free choice" and templates: how to get both at the same time*
17:45–19:00	Discussion and System Demonstration
20:00	KI99 Conference Dinner

Wednesday, 15 September 1999

10:50–11:50	INVITED TALK **Paul Heisterkamp,** DaimlerChrysler AG, Ulm: *Time to get real: Current and future requirements for generation in speech and natural language from an industrial perspective*
11:50–12:20	**Boyd Buchin, Ulf Schmerl,** Universität der Bundeswehr München: *TextPro - A Method of Generating Texts from a Formal Language into Natural Lang*
12:20–12:50	**Hercules Dalianis,** KTH and Stockholm University: *The VOLVEX Handbook - A general validation tool by natural language generation for the STEP/EXPRESS standard*
	Lunch break
13:30–14:00	**Annette Fritsch, Eric Cousin, Phillipe Tanguy,** ENST Bretagne: *A Multilingual Text Generator for Real-Time WEB-Communication*
14:00–14:30	**Sabine Geldof,** Vrije Universiteit Brussel: *Templates for Wearables in Context*
14:30–15:00	**Amanda Stent,** University of Rochester: *Content planning and generation in continuous-speech spoken dialog systems*
15:00–15:30	**Lena Santamarta,** Linköping University: *Output Generation in a Spoken Dialogue System*
15:30–16:30	Conclusions - an open discussion with all participants

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

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

July 7, 1999

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.

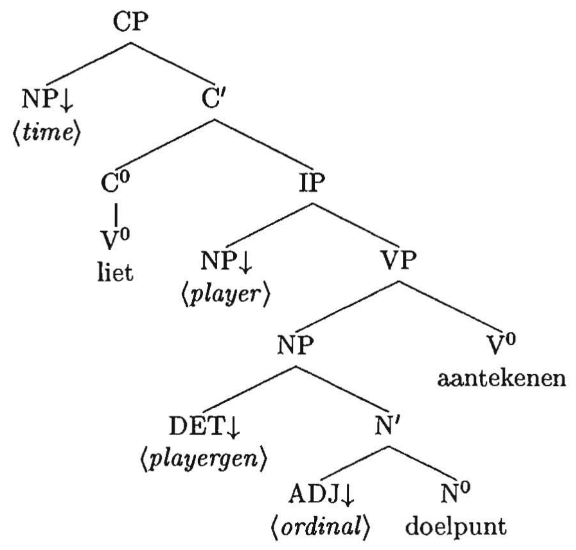
2 NLG with syntactically structured templates

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

[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".

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* ← ExpressTime (*currentgoal.time*)
player ← ExpressObject (*currentgoal.player, P, nom*)
playergen ← ExpressObject (*currentgoal.player, P, gen*)
ordinal ← 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.

```
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
```

```
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).

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 gaims) 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.

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

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.

Shallow vs. Deep Techniques for Handling Linguistic Constraints and Optimisations

Ehud Reiter
Dept. of Computing Science,
University of Aberdeen, Aberdeen, Scotland,
email: ereiter@csd.abdn.ac.uk

Abstract

An important aspect of many NLG systems is ensuring that all generated texts obey linguistic constraints and are (near-)optimal under linguistic quality measures. Where they are possible, deep techniques can automate the enforcement of linguistic constraints and optimisations. In contrast, shallow techniques require developers to explicitly enforce constraints and optimisations. Deep techniques therefore offer the potential of improving system robustness and decreasing development time. Unfortunately, deep techniques cannot be used for many types of optimisations and constraints because of gaps in our understanding of linguistic phenomena, or because the necessary software would be very expensive to create. This discussion is illustrated by examining where deep and shallow techniques are used in the STOP system, which produces personalised smoking cessation leaflets.

1 Introduction

Applied Natural Language Generation (NLG) systems should be robust, that is they should produce good-quality output in all cases, even strange situations that their developers did not anticipate. In particular, it would be very useful if we could guarantee that the output of an NLG system is always linguistically correct (correct orthography, morphology, syntax, use of anaphors, etc). In many applications, it would also be useful if we could guarantee that the output of a system was always easy to read, difficult to misinterpret, and otherwise well suited to its readers. In other words, we would like to be able to guarantee that 100% of texts produced by an NLG system obey a set of linguistic constraints (for example, are syntactically correct), and are optimal or near-optimal under a set of linguistic quality measures (for example, reading speed).

From this perspective, an important difference between shallow and deep techniques is that in systems built with shallow techniques, the system developer must explicitly ensure that constraints and optimisations are enforced by careful design and testing of templates (or whatever shallow technique is used). In systems built with deep techniques, in contrast, the core NLG code may be able to enforce some constraints and perform some optimisations automatically, without the system developer needing to explicitly worry about this. This should both enhance robustness and reduce the developer's workload.

Unfortunately, in many cases it is not possible to enforce linguistic constraints and optimisations automatically, because we do not understand the underlying linguistics well enough to be able to write a robust set of rules for the phenomena. In other cases, even if the underlying linguistics is well understood, there may not be an existing software package which can do the job, and building a deep processing engine for one application may be prohibitively expensive. In such cases, shallow techniques may be preferable.

2 STOP

In the rest of this paper I shall discuss how several linguistic constraints and optimisations are handled in the STOP system (Reiter, Robertson, and Osman, 1999)[1]. STOP produces personalised smoking-cessation leaflets, where personalisation is based on the smoker's response to a questionnaire about attitudes towards smoking, health problems, previous attempts to quit, and so forth. STOP is not fielded, but it is currently undergoing a clinical trial which requires STOP to produce 800 leaflets for previously unseen patients; hence STOP needs to be robust.

Internally, processing in STOP is divided into the three stages of document planning, microplanning, and realisation, of which document planning (deciding what information to communicate) is the most complex. Oversimplifying to some degree, the document planner works by first classifying smokers into one of 7 categories, and then activating a schema associated with that category. The schemas produce a tree-like document plan. Each leaf node of the document plan essentially defines one sentence in the leaflet. Sentences are represented by what Reiter and Dale (1999) call canned text, that is lists of sentence fragments without orthographic information such as capitalisation. The internal nodes of the tree indicate how sentences are grouped, associate document structures (such as paragraphs or itemised lists) with groups of sentences, and sometimes specify discourse relations between daughter nodes. Discourse relations are represented by cue phrases, not abstract RST-like relations (Mann and Thompson, 1988). The microplanner and realiser convert this structure into a Microsoft Word RTF document specification. STOP also includes a revision module which enforces a length constraint/optimisation (see Section 3.5); this uses importance information which the schemas associate with document plan structures. Perhaps the most innovative aspect of STOP from an NLG perspective is the knowledge acquisition methodology used to interact with experts while building the system, but this will not be discussed here.

From the perspective of deep vs. shallow handling of optimisations and constraints, I will regard as 'shallow' anything which must be explicitly programmed in a schema, and as 'deep' anything which is automatically handled by the rest of the system.

3 Handling Optimisations and Constraints in STOP

3.1 Orthography

Texts need to be orthographically correct. That is, they need to use correct punctuation and capitalisation, and should include blank spaces between tokens when appropriate. Many orthographic rules are straightforward, such as the rule that a sentence should end in a full stop or other sentence-final punctuation mark; but there are subtleties such as quote transposition (the rule in American English that if a sentence ends in a quote, the sentence-final full stop should go before the final quotation mark).

[1]More information about STOP is available at `http://www.csd.abdn.ac.uk/~rroberts/smoking.html`

In STOP, orthographic processing is handled in a 'deep' fashion, using rules based on Nunberg's (1990) analysis. This is because these rules are relatively easy to code, and (at least in my experience) it is difficult to get orthography 100% right in template-based systems, especially when a system is being developed or maintained by more than one person. Certainly most systems I have looked at which use shallow techniques for orthography do make mistakes in at least a few cases. For example, consider this output from the ECRAN system (Geldof and van de Velde, 1997):

> 'Blue Velvet' was produced by David Lynch in 1986 in the USA. The movie features Kyle McLachlan, Laura Dern, Dennis Hopper, Isabella Rossellini. It tells the story of a good but curious boy who gets in touch with evil in himself and in the world. On his 'walk on the wild side' he meets a strange nightclub singer (Isabella Rossellini), a diabolistic sadist (Dennis Hopper) and other 'strange folks'. , it will be shown at Arenberg Galeries in room 2. You can see some shots here.

The sentence *it will be shown at Arenberg Galeries in room 2* should be capitalised, and the comma in front of it should be deleted; these are orthographic errors.

Of course, I am sure the ECRAN developers could easily fix this error once it is pointed out; the point I am trying to make is simply that it is difficult for a developer to detect all such problems in advance if capitalisation, punctuation, and spacing is explicitly specified in templates. I have observed similar problems in many other systems (commercial as well as research), ECRAN is by no means atypical and I am not intending in any way to single it out for criticism.

In any case, STOP's use of deep orthographic processing seems to have been successful in its aims of making the system more robust, and of simplifying the schema author's job.

3.2 Syntactic Processing

Texts of course need to be syntactically correct, this is a very important linguistic constraint. Syntax is a complex phenomena, but it is reasonably well understood, and the NLG community has developed several general-purpose syntactic realisation packages. When we first started working on STOP, we intended to incorporate one of these packages into STOP, in order to ensure that STOP's output was always syntactically correct.

However, after experimenting with the KPML (Bateman, 1997), SURGE (Elhadad and Robin, 1997), and REALPRO (Lavoie and Rambow, 1997) realisation packages, we changed our mind and reverted to shallow techniques. That is, there is no explicit enforcement of syntactic rules in STOP; instead, schema authors must carefully design template-like structures that always produce syntactically correct text.

The problem with the packages we examined is that none of them had both adequate documentation and broad enough grammatical coverage. For example, there is essentially no documentation on the NIGEL grammar used in KPML other than a large set of examples. SURGE does have some documentation, but experimentation revealed that it has many undocumented aspects as well. For example, producing the passive form of *Sam sees Mary* (*Mary is seen by Sam*) requires not just changing the focus, but also specifying the feature (agentless no); this feature is not described in the current SURGE documentation.

Of course, it is perfectly understandable that KPML/NIGEL and SURGE should not have commercial-quality documentation. Producing such documentation is expensive, and these systems were developed as research projects, not as commercial systems. However, we felt that using a system that was not well documented might actually reduce the robustness of STOP and increase the schema author's workload.

The third system we looked at, REALPRO, was a commercial system and did have reasonable documentation. However, REALPRO's grammatical coverage did not include many constructs that we needed, and hence we could not use it in STOP. Again this is perfectly understandable; producing a well-documented commercial quality realiser is expensive, and REALPRO's grammatical coverage is dictated by what is needed in the commercial projects it is used in.

Using shallow techniques for syntactic processing in STOP was a disappointment. I hope that in the future some NLG group does develop a realisation component which is well documented, well engineered as a software artifact, and has a wide-coverage grammar; this would allow future STOP-like projects to use deep techniques for realisation.

3.3 Rhetorical Coherence

Another crucial linguistic constraint is that texts should be rhetorically coherent. A variety of 'deep' document-structuring algorithms (such as (Marcu, 1997)) have been developed which automatically create rhetorically coherent texts. These algorithms are based on formal definitions of discourse relations such as Contrast and Elaboration. We briefly considered incorporating such an algorithm into STOP, but decided against this because we felt that existing definitions of discourse relations (such as those in RST (Mann and Thompson, 1988)) were problematical. In other words, we believed that the underlying linguistic knowledge of discourse relations — what they are, when they are used, how they are expressed via linguistic mechanisms such as cue phrases — was not robust, and hence attempting to use an algorithm such as Marcu's might decrease system robustness instead of increase it.

As a result, shallow techniques were used for rhetorical coherence in STOP. Schema authors explicitly specify the order in which things are said, and also explicitly specify cue phrases between clauses or sentences. A few simple rules are enforced automatically by STOP; for example, a cue phrase will not be expressed if one of the clauses it links is the empty string. Such rules do in a small way add robustness and simplify the schema authoring task, but 95% of the rhetorical coherence task is still explicitly programmed by the schema authors.

This is not ideal, and I hope that in the future linguistic understanding of discourse relations progresses sufficiently so that general-purpose 'deep' discourse structuring systems can be built.

3.4 Reading Level

An important linguistic optimisation is that STOP's texts should be easily readable by people with limited reading ability; in other words, that STOP's texts should have a low 'reading level'. Unfortunately, there is no reliable measure of the reading level of a text (although there are some rough heuristics, such as the Flesch Reading Ease score (Hartley, 1994)). However, there are some guidelines on ways of decreasing readability level, such as using short sentences, short familiar words, and active voice; and avoiding sentences with more than two subordinate clauses (Hartley, 1994). These can in principle be implemented in a deep fashion, even if it is not possible to measure overall reading level.

In STOP, this optimisation is handled in a shallow manner, that is schema authors explicitly specify words and sentence structures which are expected to be in accordance with the above rules. This decision was partially based on the way we interacted with our reading-level expert; she preferred to revise specific sentences, and this was easiest to do if sentence fragments were explicitly represented in schemas. However, we are currently trying to see if we can automatically enforce some of the above rules in STOP in a deep fashion.

3.5 Length

An important application-specific constraint/optimisation in STOP is length. Essentially, STOP's leaflets must fit on 4 A5 pages (a constraint), but we wanted them to say as much as possible given this constraint (an optimisation). This is enforced in a deep fashion in STOP, using a revision module which estimates the length of a leaflet and adjusts content accordingly. This was useful because length constraints/optimisations are difficult for schema authors to enforce explicitly (essentially because they are global, not local), so automating this both enhanced robustness and made the schema authoring job easier.

3.6 Other Constraints and Optimisations

The constraints and optimisations discussed above are all important and non-trivial to enforce in STOP. Hence, the decision as to whether they should be automated or left to schema authors depended on the issues raised at the beginning of this paper. There were other constraints and optimisations which were handled by shallow techniques simply because it was very easy to enforce them in schemas, or because they were not important in STOP. For example, morphological constraints (such as the correct formation of plurals and other inflected forms), could be automated in a deep fashion, but this was not done in STOP because there were very few places in STOP's leaflets where inflected forms needed to be produced from root words, and hence correct morphology was easy to directly specify in schemas. Another example is recall optimisation (that is, optimising the amount that recipients remember after reading a leaflet); this was felt to be unimportant in the STOP application, and hence no attempt was made to optimise this (either automatically or by the schema authors).

4 Conclusion

In conclusion, deep and shallow techniques differ in how they ensure that linguistic constraints and optimisations are enforced. In an ideal world with perfect understanding of language and unlimited software development resources, deep techniques would be used everywhere, because in principle they are better at guaranteeing that 100% of generated texts satisfy linguistic constraints and are (near-)optimal under linguistic quality measures. However, in our imperfect world of limited understanding of language and limited resources, sometimes it makes more sense to use shallow techniques.

STOP uses deep techniques where we felt that (a) they addressed a linguistic constraint or optimisation which was important in STOP, and could not trivially be enforced by schema authors; (b) the underlying linguistics of the constraint or optimisation was well understood; and (c) deep processing could realistically be implemented in a resource-limited project. Where we did use deep techniques (orthography, length, a few rhetorical coherence rules), we believe they added substantial value to the system in terms of making it more robust and easier to develop. However, there were many areas where we could not use deep techniques because of insufficient understanding of the underlying linguistic phenomena, or the expense of programming a robust implementation of a deep technique.

We hope that future systems such as STOP will be able to make more use of deep techniques, because of advances in linguistics and the development of reusable wide-coverage NLG components that are robust, well documented, and well engineered as software artifacts. After all, much of the the potential power of NLG technology comes from using deep techniques to automatically handle linguistic constraints and optimisations. Indeed, without such techniques, we may not be able to do much better than developers who build text-generation systems using string-concatenation or mail-merge techniques.

References

Bateman, John. 1997. Enabling technology for multilingual natural language generation: the KPML development environment. *Natural Language Engineering*, 3:15–55.

Elhadad, Michael and Jacques Robin. 1997. SURGE: A comprehensive plug-in syntactic realisation component for text generation. Technical report, Computer Science Dept, Ben-Gurion University, Beer Sheva, Israel.

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

Hartley, James. 1994. *Designing Instructional Text.* Kogan Page, London, third edition.

Lavoie, Benoit and Owen Rambow. 1997. A fast and portable realizer for text generation. In *Proceedings of the Fifth Conference on Applied Natural-Language Processing (ANLP-1997)*, pages 265–268.

Mann, William and Sandra Thompson. 1988. Rhetorical structure theory: Towards a functional theory of text organisation. *Text*, 3:243–281.

Marcu, Daniel. 1997. From local to global coherence: A bottom-up approach to text planning. In *Proceedings of Fourteenth National Conference on Artificial IntelligenceAAAI-1997)*, pages 629–635.

Nunberg, Geoffrey. 1990. *The Linguistics of Punctuation.* Number 18 in CSLI Lecture Notes. University of Chicago Press.

Reiter, Ehud and Robert Dale. 1999. *Building Natural Language Generation Systems.* Cambridge University Press. In press.

Reiter, Ehud, Roma Robertson, and Liesl Osman. 1999. Types of knowledge required to personalise smoking cessation letters. In Werner Horn et al., editors, *Artificial Intelligence and Medicine: Proceedings of AIMDM-1999*, pages 389–399. Springer-Verlag.

From full generation to 'near-templates' without loosing generality

John Bateman

University of Bremen
Bremen, Germany

e-mail: bateman@uni-bremen.de

Renate Henschel

Human Communication Research Centre,
University of Edinburgh

e-mail: henschel@cogsci.ed.ac.uk

1 The state of the 'art'?

Natural Language Generation (NLG) research has traditionally sought after sophisticated complexity of expression in the texts generated and considerable degrees of flexibility. However, the applications involving NLG functionality attempted to date have required little of this flexibility and none of the complexity. In this rather immature context, a received view is developing that full NLG cannot be motivated for 'real applications' and that simpler techniques, such as 'template generation' should be used. But, since templates in the sense of rigid patterns are well acknowledged to be overly restricted, 'extensions' of basic templates are commonly proposed: ranging from libraries of fixed syntactic fragments (Glass, Polifroni & Seneff 1994), through 'phrasal lexicons' (e.g., Milosavljevic, Tulloch & Dale 1996), through to general mechanisms for constructing grammars involving arbitrary chunks of linguistic material (Busemann 1996). The 'application context' assumed by all of these directions is arguably appropriate for the current state of NLG-application: i.e., generation-like behaviour of restricted flexibility is to be produced making little or no use of existing resources, with no conformance to existing standards, with little expectation that this is any more than a one-off effort, and with minimal regard for possible extension.

Since this kind of situation is untenable for anything more than a handful of first experiences in providing generation functionality, template-based environments such as that of Busemann (1996) offer powerful techniques of which 'template-grammars' can avail themselves as little or as much as they require. It is possible then to write 'template-grammars' of increasing sophistication, largely overcoming many of the problems of rigidity commonly attributed to them. The argument is then that such developments are useful in that they focus on the needs of applications rather than on the interests of theoretical generation; as, for example, Horacek & Busemann (1998) claim:

> '... most available tools are based on *in-depth* approaches to NLG contributing to a general purpose generation system rather than supporting the economic development of dedicated applications.'

In this paper, we will suggest that the dichotomizing of approaches to NLG that such views promulgate is unnecessary and that appropriate approaches to generation can move freely between the general and the specific. 'In-depth' does not mean that there is no basis for pursuing economic solutions—indeed, quite the contrary, a premature orientation to 'shallow' techniques can weaken support for the economic development of increasingly complex, but nevertheless practical, generation systems while simultaneously stifling opportunities for user-motivated (even user-demanded) flexibility.

It is now commonly noted that the distinction previously drawn between 'full' generation and template-based generation is not clear-cut; indeed, the two approaches need not be considered different in kind at all, but instead represent two extreme points on a continuum. This has supported the development of 'hybrid' or 'mixed' architectures for generation, in which some elements of a generator's behaviour are contributed by relatively fixed templates and others by full generation. Both views—that template generation and full generation are extremes on a single continuum and that mixed representations are possible—are natural when a unification-based metaphor is assumed for representing linguistic resources and their processing. At some intermediate point in processing some of the structures constructed will be partially instantiated and resemble more templates, others will be uninstantiated and resemble more the state of affairs in full generation.

Now, while the removal of the categorial distinction between full-NLG and template-NLG has enabled template-based approaches to address issues of flexibility and potential expansion, it equally enables full-NLG to address issues of application-customization. Just because an NLG system *can* provide in-depth approaches, does not mean that it *must* in all cases. Moreover, while template-based approaches make much of their ease of adaptability—i.e., that a developer is free to write whatever they need to get desired generation functionality with minimal external constraint and with no necessary adherence to existing practice—the downside of this is that issues of standardization and re-use get left by the wayside. A solution to this is relatively obvious: libraries of established solutions to particular areas of generation functionality should be made available for the particular template/hybrid-architecture adopted so that these modules can be used or not when building generation functionality for some application. But one thing that should be clear in this move is that the apparent 'freedom' of the wild west of template generation is successively replaced by a more disciplined adherence to, and use of, standard packages and standard techniques and this will necessarily represent an initial overhead for an application builder.

It is this, then, rather more complex situation of application development that needs to be compared and contrasted among other approaches to providing NLG-technology for applications. The argument that some full-NLG approach is complex and involves an overhead when compared with 'quick and easy' template generation can only apply for the most simple of generation requirements (generally those for which it is difficult to motivate NLG-technology in the first place). For more complex, more mature application demands, *any* appropriate solution will involve external overheads: that is, overheads due both to the complexity of the system that is used and to the need to re-use and conform to pre-existing resource components. The question is then to what extent these overheads can be structured in beneficial ways.

2 Automatic customized subgrammar extraction

The view of practical NLG given above is largely 'bottom up': when particular generation functionality is required, various resource components are either constructed from scratch or adopted from any libraries that may exist. We can contrast this with another view, where the developer works with the power of a full description which is then subsequently automatically pruned of non-required functionality in order to deliver a small, customized application system. Solutions of this kind are generally not quite so small or fast as those delivered by the former model (arguably), but are (again arguably) very much more rapid to develop and retain a degree of flexibility and ease of subsequent extension that the former model cannot match. In Henschel & Bateman (1997), we introduced and illustrated an implemented method for systematic, semi-automatic customization of this kind. Here we describe its use for a different application and different language, focusing on some of the options that having a full NLG system 'in the background' open up.

Although the procedure we have developed is valid for grammars written in typed unification formalisms in general, the principal reference grammars we consider are systemic-functional generation grammars. Large-scale systemic grammars have shown themselves to be powerful tools for a wide range of generation tasks. Computational instances of systemic grammars are accordingly employed in some of the largest and most influential text generation projects—such as, for example, PENMAN, TECHDOC, Drafter-I, Gist, and the numerous projects using SURGE/FUF (Elhadad & Robin 1996). The general methodology adopted in these systems has been to build on the already broad coverage systemic-functional grammars that have been constructed over the past 15 years. Large grammars here include the original Nigel grammar for English developed within the Penman text generation project (Mann & Matthiessen 1985) and SURGE. The input specifications for these resources are usually a semantic specification of some kind: for the purposes of the current paper we will focus here on Penman-style systems since their favoured input is the 'Sentence Plan Language' (SPL) as generally proposed as a reference input form for tactical generation by the RAGS ('Reference Architecture for Generation Systems') project (RAGS Project 1999).

Whereas the increment involved in extending these large grammars to cover any particular phenomena required by a new application or text type is usually relatively small (i.e., much less than building comparable functionality from scratch), maintaining the resources as a whole would be a considerable overhead for most applications. Henschel & Bateman (1997) therefore describe an algorithm and its implementation by which arbitrary ***subgrammars*** can be automatically extracted from the resource set as a whole in order to provide customized generation capabilities. The input to the extraction process is a set of grammatical types to be preserved as distinct in the extracted subgrammar. Such a list is most simply obtained by generating the required output with the full grammar. Tuning to a particular application then works by (a) constructing (or planning) semantic inputs for the tactical generator, (b) generating the surface strings required for the application, (c) collecting the subset of grammatical types involved, and (d) producing a subgrammar that is customized so as to just cover the target generation behaviour.

3 Case study: temporal specifications for appointments

We illustrate the approach by considering an appointments domain and appointment scheduler such as described in the COSMA-system (Busemann, Declerck, Diagne, Dini, Klein & Schmeier 1997). We begin our illustration by assuming that such an appointment scheduler has to generate a limited number of speech acts that could be represented by templates such as '<meeting> has been postponed <time>', etc. Now, temporal expressions should be considered a standard part of a full generation grammar and we want to *re-use* this information for our targetted application. For the concrete generation grammar Nigel and a generation system such as Penman or KPML (cf. Bateman 1997), temporal expressions are generated by purely semantic SPL specifications: the following SPL, for example, would generate (in English) the string 'until Monday'.

```
(time / object
   :modifying-relation-q modifying
   :operator-id (s / (extremal anterior))
   :operand-id (d / date :weekday-id (wd / monday)))
```

Here `operator-id` describes the semantic type of the temporal relation involved, and `operand-id` describes the semantics of the prepositional object. This could then be embedded in the 'hybrid' SPL-style supported by KPML so as to give an input specification:[1]

```
((p / template :pattern (event "has been postponed" time))
 (time / object
    :modifying-relation-q modifying
    :operator-id (s / (extremal anterior))
    :operand-id (d / date :weekday-id (wd / monday))))
```

Although the use of a template avoids much of the complexity of the full Nigel grammar, it would still not be desirable for the simple appointments domain to keep the full grammar 'on board'. Even the grammar component responsible solely for prepositional phrases of this kind contains approximately 50 choice points with cross-classification, while the grammar of nominal phrases necessary for the prepositional objects adds another 80 or 90 choice points. Together this involves several hundred grammatical types. If we then further assume that we actually only need to generate very few phrases, e.g., the phrases 'until <time>', 'by <time>' and 'on <time>' then the full grammar is clearly overkill.

In constrast, automatic grammar extraction allows us straightforwardly to extract a subgrammar that corresponds to just the breadth of generation that we require. When we base extraction on the semantic specifications necessary for the three temporal expressions we need, the result is a 'complete' generation resource with only a handful of choice points, presenting little more complex than a template with conditionalized components; this is shown in Figure 1. Moreover, the subgrammar extraction process maintains an appropriate link between the semantics and the extracted subgrammar: this means that the input specifications are *unchanged* and can continue

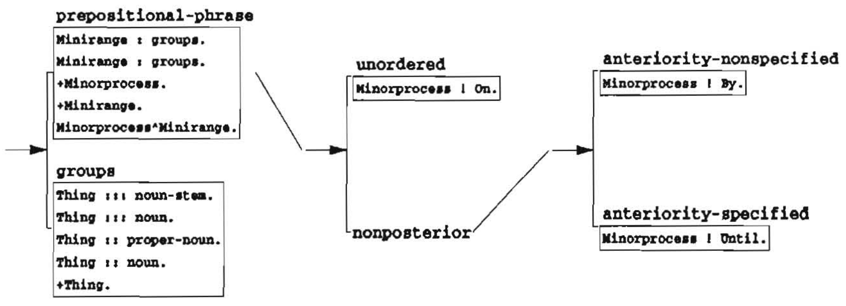


Figure 1: Automatically extracted grammar for simple temporal specifications

to be drawn from stable libraries of test suites and examples, as well as providing a stable API for applications.

Although it is possible to proceed in the fashion just described, some of the well-known limitations of the template approach show themselves when we switch the language. We could, for example, just as easily select to send the semantic specification given above through the existing grammar for German. Then we would obtain the not very useful string: 'Event has been postponed bis Montag'. Language-conditionalization of the input specification is supported in KPML but this does not avoid the basic problem that the particular appropriate choice of preposition depends on what the system intends to communication: when it is 'postponing' (*until*, *auf*) rather than 'taking-place' (*on/at*, *an/um*) particular decisions are necessary and this depends on information that is not available in the template. In short, the apparent simplicity of the approach brings complexity downstream because general linguistic information that a full grammar possesses is not accessible.

As a more complex example, we can also consider the case where we want to generate not just three simple temporal expressions but a representative range of the expressions that actually occur in an appointments domain. For this, we used the corpus of German email messages collected prior to the COSMA project (Declerck & Klein 1997): this includes expressions such as the following in many combinations:

$$\left\{\begin{array}{c} \text{14} \\ \text{14h} \\ \text{14:00} \\ \text{um 14:00 Uhr c.t.} \\ \text{um 14:00 s.t.} \\ \text{um 2} \\ \text{um 14:00h} \\ \text{um 14 Uhr} \end{array}\right\} \left\{\begin{array}{c} \text{4.11.91} \\ \text{4. November 1991} \\ \text{4. November 91} \\ \text{4.11} \\ \text{04.11} \\ \text{4. November} \\ \text{November} \\ \text{November 1991} \end{array}\right\}$$

We extended the coverage of our German grammar so that all of these expressions can be generated on the basis of semantic SPL specifications. Since many of the variations have immediate and restricted linguistic possibilities, the resulting SPLs are often not significantly different from a keyword-based input form—this can be seen in the following two examples:

14:00 Uhr c.t.

```
(t / clock-time
   :academic-time-q academic-time
   :hour (y / object :name |14|)
   :min  (m / object :name |00|))
```

14.00 s.t.

```
(t / clock-time
   :academic-time-q academic-time
   :academic-time-type-q exact
   :clock-explicitness-q clock-not-expressed
   :hour (y / object :name |14|)
   :min  (m / object :name |00|))
```

Such input expressions can be used with the full grammar anywhere where corresponding temporal expressions are to be generated. The full German grammar then contained 719 choice points

[1]The event that is postponed could also be a template or a full semantic specification of some event.

ranging over 1200 grammatical features.

We then produced a set of input expressions involving the full range of temporal expressions and covering the kinds of speech acts typically found in the corpus (e.g., postponing meetings, moving meetings forward, announcing meetings for particular dates, arranging to meet on some date, etc.). As with most real examples of the use of this style of generator, the values of many features are provided by default values and do not need to be explicitly given. Therefore sentences such as: "<event> wird auf <time> verschoben" are produced from SPL patterns of the following form, where <time> can be drawn from any of the temporal specifications summarized above:

```
(x27 / dispositive-material-action
  :lex postponing
  :tense present
  :actee <event>
  :destination <time>)
```

Now sufficient information is explicitly available in the input specification, the grammar can decide between particular prepositional forms appropriately—and, moreover, this then does not present a problem when the language changes.

Finally, we ran this set of semantic specifications through the grammar extraction process in order to derive a time-specification customized generation grammar. The specifications as a whole employ 308 of the grammatical features of the full grammar when generating and so an extracted grammar can be substantially smaller: in this case, the extracted grammar consisted of only 139 choice points ranging over 204 grammatical features. We then regenerated the example set with the extracted grammar and also compared generation times. Whereas generation with the full grammar ranges from 1–4s with an average just below 2s, generation with the extracted grammar requires approximately 60% of that.[2]

In fact, the range of expressions generated with our example set is still probably too wide for a generation system. It is not likely that an application would need to produce all the variability illustrated above. However, since the time and date specifications now form part of the general grammar, more specific, application-tuned subgrammars can be similarly extracted. All that needs to be done is that a system designer would pick the particular forms of temporal expressions that are required, and grammar extraction can deliver the result, ranging from the trivial component illustrated at the beginning of this section to the more complex coverage of the last example. This also suggests both useful support, and flexible production of resources, for controlled languages.

4 Discussion

The brief example of the previous section has suggested using hybrid template-semantic input forms, applying existing linguistic resources unchanged for a particular domain, and the automatic extraction of subgrammars that range from being little more complex than templates to sizable, although application-motivated, grammars of their own.

Several issues are worth raising. In particular, we have assumed throughout the present discussion that an input form analogous to SPL is appropriate. This could well vary depending on particular application: however, given the above mentioned proposal from the RAGS project that SPL be seen as a reference point, it would be useful to develop libraries of SPL specifications for a wide variety of grammatical constructions—this would provide a strong basis for applications to aim at when generation functionality is required. Appropriate input specifications can then either be generated as traditionally done in NLG or used as hybrid 'semantic templates' as illustrated above. Note that this automatically provides a convenient point for cross-language localization. Another issue is the kind of division of labour our methodology requires: whenever there is generation capability that is not present in a general resource, we have argued that the appropriate place

[2]All times with Allegro Common Lisp 5.0 running on a Sun Ultra 2. Sentence length ranges from 5 to 18 words, average 10 words. No further performance improving measures were taken in the comparison and generation was carried out with the unoptimized full grammar development environment.

to build this capability is in the general resource (so that it can be re-used) and not in the specific application. Both options are probably desirable, however: minor adaptations being carried out in the extracted grammar, with more significant areas being added to the general grammar. Finally, our approach places a different slant on 'integration': with the grammar extraction model, integration of linguistic information is done within the general grammar rather than opportunistically as required when particular components are to be brought together for an application. This additional effort is then well repaid in the flexibility of customization and tailoring that becomes possible.

To conclude, we suggest that modularities and resource re-use such as those envisaged here can provide a basis for the more mature application work involving generation that will be required in the future and will support a more useful utilization of the mutually-beneficial competences of linguistic and software engineers.

Acknowledgement

We are very grateful to Stephan Busemann for making the appointment expression corpus available to us.

References

Bateman, J. A. (1997), 'Enabling technology for multilingual natural language generation: the KPML development environment', *Journal of Natural Language Engineering* **3**(1), 15–55.

Busemann, S. (1996), Best-first surface realization, *in* 'Proceedings of the 8th. International Workshop on Natural Language Generation (INLG '96)', Herstmonceux, England, pp. 101–110.

Busemann, S., Declerck, T., Diagne, A. K., Dini, L., Klein, J. & Schmeier, S. (1997), Natural language dialogue service for appointment scheduling agents, *in* 'Proc. 5th Conference on Applied Natural Language Processing', Washington, DC., pp. 25–32.

Declerck, T. & Klein, J. (1997), Ein email-korpus zur entwicklung und evaluierung der analysekomponente eines terminvereinbarungssystems, *in* 'Proceedings of DGfS-CL', Heidelberg.

Elhadad, M. & Robin, J. (1996), A reusable comprehensive syntactic realization component, *in* 'Demonstrations and Posters of the 1996 International Workshop on Natural Language Generation (INLG '96)', Herstmonceux, England, pp. 1–4.

Glass, J., Polifroni, J. & Seneff, S. (1994), Multilingual language generation across multiple domains, *in* 'Proceedings of the International Conference on Spoken Language Processing', Yokohama, Japan.

Henschel, R. & Bateman, J. (1997), Application-driven automatic subgrammar extraction, *in* 'Proceedings of ACL/EACL97 Workshop: "ENVGRAM: Computational Environments for grammar development and linguistic engineering', Association for Computational Linguistics. (Also available from the Computational Linguistics E-Print Archive, paper: cmp-lg/9711010).

Horacek, H. & Busemann, S. (1998), Towards a methodology for developing application-oriented report generation, *in* A. Günter & O. Herzog, eds, '22nd German Conference on Artificial Intelligence (KI-98). Proceedings', Bremen, Germany.

Mann, W. C. & Matthiessen, C. M. (1985), Demonstration of the Nigel text generation computer program, *in* J. D. Benson & W. S. Greaves, eds, 'Systemic Perspectives on Discourse, Volume 1', Ablex, Norwood, New Jersey, pp. 50–83.

Milosavljevic, M., Tulloch, A. & Dale, R. (1996), Text generation in a dynamic hypertext environment, *in* 'Proceedings of the Nineteenth Australasian Computer Science Conference (ACSC '96)', Melbourne, Australia.

RAGS Project (1999), Towards a reference architecture for natural language generation systems, Technical Report ITRI-99-14 and HCRC/TR-102, Information Technology Research Institute (U. Brighton) and Division of Informatics/Human Communication Research Centre (U. Edinburgh), Brighton and Edinburgh. Contributors: Lynne Cahill, Christy Doran, Roger Evans, Chris Mellish, Daniel Paiva, Mike Reape, Donia Scott and Neil Tipper.

"Free choice" and templates: how to get both at the same time

Jo Calder[1,3], Roger Evans[2], Chris Mellish[3], and Mike Reape[3]

[1] Human Communication Research Centre
University of Edinburgh
[2] Information Technology Research Institute
University of Brighton
[3] Institute for Communicating and Collaborating Systems
Division of Informatics
University of Edinburgh

Abstract. This paper presents a procedurally neutral framework for representing the results and states of computations called the *whiteboard*, developed in the context of an investigation of reference architectures for Natural Language Generation (NLG) systems. We show that whiteboards solve a number of data representation problems in NLG, in particular, how to characterize *partial* and *mixed* representations. With these in place, an approach to generalized "canning" — by which we mean the generalisation of its use in "canned text" to any datatype — becomes available which allows inclusion of arbitrary fixed and partial structures of any type which may themselves be realized by structures of other types. We use this mechanism to show that the "free choice vs. templates debate" is only a question of degree.

1 Introduction

The context of the work described in this paper is an initiative to develop a generic architecture for NLG systems[1]. In a study of applied NLG systems (reported in [Pai98] and [CDE$^+$99b]), we found reasonable evidence to support a functional decomposition of most systems into seven or so modules operating on three types of data representation (broadly, rhetorical, semantic and syntactic - see [CDE$^+$99a] for further details). However, we found less consistency in ordering and control of processing within each system, leading us to conclude that any generic architecture would need a high degree of procedural flexibility.

This paper outlines an architectural framework that aims to support this flexibility. We provide an infrastructure that allows modules in an NLG system to communicate with each other and collectively develop data structures that represent solutions to a generation task. We focus on the declarative aspects of the framework, encompassing both data structures of the generation task and relationships between such structures. In particular, our proposals naturally support structures that span more than one representation level ('mixed' structures), partial structures and 'canned' structures of any type. We do not address issues of control directly, but intend the declarative semantics of the framework to support any control regime. The framework we have developed has the following key properties:

- It is *cumulative*. That is, there is a data stream which flows from the beginning of the generation process to the end and once data is added to this stream, it stays there - it cannot be removed or altered (although it may be superseded by another data item of the same type).
- It represents data as typed atomic objects and relationships between them - objects may have internal structure accessible to individual processing modules, but apart from the object's type, that structure is invisible at the level of the framework itself.
- It supports a range of control regimes (such as incremental, revision, parallel and blackboard regimes). To do this, it can explicitly represent results of intermediate and partial representations, and realizational and historical dependencies between data objects.

We call this representation framework a *whiteboard* architecture[2].

[1] This work was supported in part by ESPRC grants GR/L77041 (Edinburgh) and GR/L77102 (Brighton), RAGS: Reference Architecture for Generation Systems, and research sponsorship from Microsoft Corporation to Edinburgh.

[2] We choose the term *whiteboard* in contrast to so-called *blackboard* architectures. A crucial property of blackboards is that their contents can be removed or destructively modified. A whiteboard, by contrast, can only be written to - once added, no content can be removed or altered.

2 Whiteboards

Informally, a whiteboard is an unstructured heap of (undecomposed) objects with a structure imposed on them. The structure we want to impose in the general case is one of stating that two elements in the heap stand in some relationship to each other.

This requirement can be achieved with the following simple model. We have a set of objects and a set of arrows pointing between pairs of objects. Both the objects and the arrows are *typed*: there will be a function from objects to object types, and arrows to arrow types. In the case of arrows, there is also a second level of typing. Each arrow type, t, has a ***signature***, $\sigma(t)$, which is a pair of object types, indicating what type of object arrows of type t can point from and to. We define a whiteboard as follows.

Definition 1 (Whiteboard).
A whiteboard *is a tuple*

$$\langle O, A, T_O, T_A, t_O, t_A, \sigma\rangle$$

where O is a set of objects, *$A \subseteq (O \xrightarrow{T_A} O)$ is a set of of* arrows, *T_O is a set of* object types, *T_A is a set of* arrow types, *$t_O : O \rightarrow T_O$ is a function, $t_A : A \rightarrow T_A$ is a function, $\sigma : T_A \rightarrow 2^{(T_O \times T_O)}$ is a function and for all $a : o_1 \xrightarrow{t} o_2 \in A.(\langle t_O(o_1), t_O(o_2)\rangle \in \sigma(t))$.*[3]

By intention, nothing is said in the definition about the internal structure of objects or the internal structure of types. We assume that object types "have their own logic" which is to be strictly separated from the information encoded in the whiteboard. This is because we want the whiteboard to be independent of any particular set of objects. That is, a whiteboard should make sense no matter what its set of objects is. However, we shall assume below that types corresponding to sets and tuples of objects are available, so that the whiteboard can contain objects representing sets and tuples of other objects.

This strategy induces a very explicit division in the representation of information in the system as a whole. The whiteboard represents information at the level at which modules communicate with each other. In particular, modules are only sensitive to 'events' in the whiteboard at this level of representation. Lower levels of representation may also exist *within* whiteboard objects (for example a whiteboard object might be a feature structure), and modules may pass information between each other at this level, but such structure is invisible to the whiteboard.

Note however, that our formulation takes absolutely no stand on the 'granularity' of the whiteboard objects, that is *where* this division in representations occurs. The whiteboard might characterize just the highest level architecture where each component operates on richly structured representations whose internal structure is invisible to the whiteboard, or at the other extreme the whiteboard could record every use of the lowest level data access or constructor functions.

A further issue on which the definition is intentionally uncommitted is the extent to which there might be different *kinds* of arrows. In practice it is often useful to distinguish at least two such kinds, namely arrows that relate pieces of structure together to form larger structures (for example, linking mothers to daughters in a phrase structure tree) and arrows that relate whole structures to each other (for example linking semantic structure to syntactic structure). Although this distinction is not precise, it is useful to introduce some terminology to capture the intuition: let us call the first kind of arrows *local* and the second *non-local*. The following examples of arrow types are significant for the discussion below.

constructor (local) constructor arrows build new objects out of old — the arrow points from the substructure to the structure it is part of.
component (local) component arrows are (for the purposes of this paper) just the inverses of *constructor* arrows.
realizes (non-local) the target is the next level of representation (derived from the source) in the text generation process.

3 Coherence and completeness

In this section we briefly note some formal properties of whiteboards. Let us assume we express the components of a generation system in terms of functions F over the objects O of a whiteboard. As elements on the whiteboard can correspond to sets or tuples of objects, so we can view all functions as unary. Let $\sigma(f)$ be the signature of the function f.

[3] This simple setup might be profitably generalized to an order-sorted type structure.

Definition 2 (Coherence). *A whiteboard is* coherent *iff* $\forall f \in F, \forall o, o_1, o_2 \in O.(f(o) = o_1$ *and* $f(o) = o_2) \rightarrow (o_1 = o_2$ *and* $\exists a \in A.a : o \xrightarrow{f} o_1)$.

Definition 3 (Completeness). *A whiteboard w is* complete *iff w is coherent and* $\forall f \in F, o \in O.(t_O(o) = \sigma(f) \rightarrow \exists o' \in O.f(o) \xrightarrow{f} o' \in A)$.

In other words, coherence requires that functions really are functions whose 'outputs' are recorded on the whiteboard. Completeness requires that every function "has been applied" to every object in its domain.[4]

The following definition is also useful. An object o is *initial*, if it is not the result of any function. Then, given any set of initial objects, and assuming a condition of minimality in which the whiteboard contains only initial objects and objects transitively derived by function from them, the induced minimal complete whiteboard is unique.

These definition provides the basis for viewing a whiteboard as the basis of a processing system. The whiteboard is seeded with some initial objects, and then processing modules operate on the objects in the whiteboard, adding further objects and relationships. The process continues until no further objects can be added. As long as each processing module preserves coherence then the resulting whiteboard will be the unique minimal complete whiteboard.

As a cumulative data store, the whiteboard resembles the chart as used in many parsing algorithms. Indeed, work by [Ritar] on determining properties of grammars via the notion of 'completed' charts has been a major inspiration to us. In a setting where productions in a CFG may be restricted to top-down (or bottom-up) use, he derives conditions under which completeness with respect to the unrestricted grammar is preserved. As is well known, an edge in a chart can in fact represent the 'result' of more than one computation and these computations are usually distinct with different categories and substructure assigned. We attempt here to mimic exactly this kind of behaviour in a more general setting.

We will say that an object *has a history* if it is the target of some *realizes* arrow. In other words, it may be dependent on one or more other objects, but is not the realization of any of them. Initial objects don't have histories.

4 Some examples

An example of a *realizes* arrow might be $x \xrightarrow{\mathsf{syn}} y$ which means that y is the syntactic structure that realizes x. Then if we already have a semantic object a and we have just created a syntactic object b which realizes it then we just add b to the whiteboard and add an $a \xrightarrow{\mathsf{syn}} b$ arrow. Likewise, we might under some circumstance imagine adding a semantic object a and an arrow $\xrightarrow{\mathsf{syn}}$ pointing from a to a syntactic object b which was already there.

We might also want to say that the syntactic object which realizes some semantic object a is the same thing as one of the components of some other syntactic object, b without knowing anything about what that object is. In this case, we add a new object, say c, and a *realizes* arrow from the semantic object a to c and a *component* arrow from the syntactic object b to c. In this case, we leave any component structure of c undefined. Procedurally speaking, when we eventually get around to computing any component structure of c we will add arrows to characterize that structure. (Cf. Fig. 1.[5])

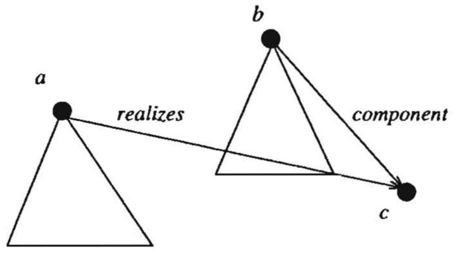


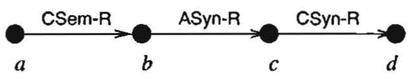
Fig. 1. An example of an object with an unspecified type

[4] An obvious technical consequence is that all functions in F are total on their domains. This is easily arranged.
[5] The triangles in Fig. 1 and the following diagrams indicate objects with internal structure.

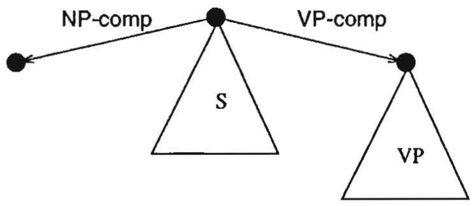
5 Partial and mixed structures

Fundamentally, a whiteboard is just a collection of objects and arrows. However our informal distinction between local and non-local arrows allows us to conceive of larger structures within a whiteboard. A *local structure* (or just 'structure') is a maximal connected subgraph in which all the arrows are local. Local structures correspond to 'ordinary' data structures[6].

Intuitively, a *partial structure* is a structure with "a hole in it". For example, a syntactic representation might represent an NP-VP clause with the VP syntax fully specified, the mother S fully specified but a hole where the NP goes. In this case the object corresponding to the S will have a *component* arrow from it to the syntactic object corresponding to the VP but the *component* arrow for the NP will just point to some object with an "underspecified" type. (Cf. Fig. 2(b).)



(a) A mixed structure



(b) A partial structure

Fig. 2. Structures

A *mixed structure* is a set of local structures connected by non-local arrows. For example, consider a generation system with four levels of representation: A(bstract)Sem(antics), C(oncrete)Sem(antics), A(bstract)Syn(tax) and C(concrete)Syn(tax) and that the architecture is one of a pipeline of transducers with the four levels of representation as interfaces in that order.[7] Then, a mixed structure might corresponds to an object and some of the objects which were transduced to generate it. That is, if we have $a \in \mathsf{ASem}$, $b \in \mathsf{CSem}$, $c \in \mathsf{ASyn}$ and $d \in \mathsf{CSyn}$ and arrows $a \xrightarrow{\mathsf{CSem-R}} b$, $b \xrightarrow{\mathsf{ASyn-R}} c$ and $c \xrightarrow{\mathsf{CSyn-R}} d$ where CSem-R, ASyn-R and CSyn-R are the "realizes CSem", "realizes ASyn" and "realizes CSyn" arrow types respectively, then we can look at the collection of a, b, c and d as a mixed structure. (Cf. Fig. 2(a).)

Notice however that any of the structures in Fig. 3 would be a valid mixed structure as well. In these cases, levels of representation are "skipped" and objects of one level of representation directly realize objects of another level of representation without all the objects of the intermediate levels of representation present.

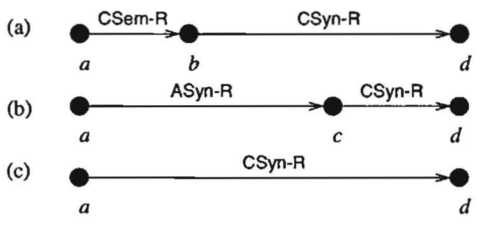


Fig. 3. Some other mixed structures

For example, in Fig. 3(a), the ASem object a is conventionally realized by the CSem object, b, but b is not realized by an ASyn object but is instead realized directly by the CSyn object d. This might be the case for a number of reasons. Two are that d may have been computed directly from b or perhaps for some reason the intermediate ASyn object has simply not been put in the whiteboard. Fig. 3(b) and

[6] Strictly, we need to identify a unique 'root' object to ensure this – a local structure with multiple roots corresponds to a *tangled* structure: two or more objects that share components. But we will not pursue this detail here.

[7] These levels are among those proposed within the RAGS project, [CDE+99a]

Fig. 3(c) are similar. In Fig. 3(b), the CSem object is missing and the ASem object a is directly realized by the ASyn object c. In Fig. 3(c), both the CSem and ASem object a is directly realized by d.

If an object is added without its derivationally precedent levels of representation as in Fig. 3, computation can still proceed with the object as long as access to those preceding levels is not required. For example, an ASyn object might be placed in the whiteboard and then a CSyn object calculated from it so long as realization does not require access to precedent ASem and CSem objects. But this is just the behaviour required. Some computations will require derivationally precedent levels of representation and some will not. We will exploit this property below.

Continuing our analogy with chart parsing, once an object arrives on the whiteboard, its contents are available for further processing regardless of the history of that object. Once a set of objects is available which provides the argument required by some function, application of that function is then a possible computation step. It could be that some collection of objects are processed in many different ways, just as edges in a chart may be used by more than one dominating edge.

6 "Free choice" vs. templates

We have described how the whiteboard allows us to represent local, mixed and partial structures in a uniform framework, and also introduced the notion of object histories. In this section we put these components together to give an account of *generalised canning*, arbitrary mixed canned representations. First, we define *canning*.

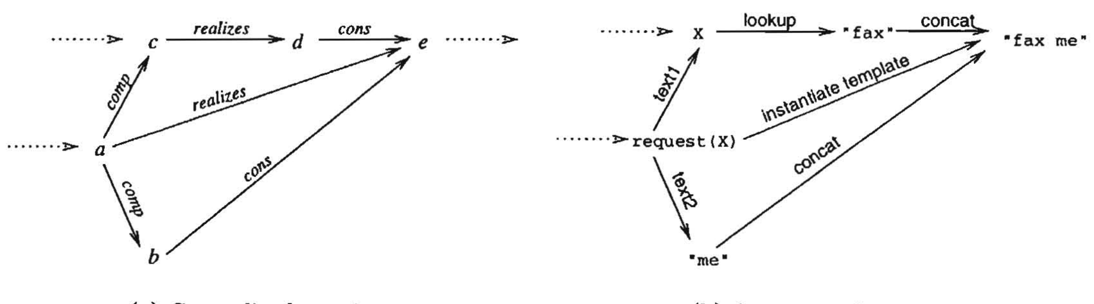
Definition 4 (Canning). *An object o in a whiteboard is* canned *if it represents a data-type which could potentially be constructed by some function, but its contents do not have a history.*

It is worth emphasizing that this definition operates in terms of *types*, rather than instances.

As a a concrete example, consider a traditional template consisting of predetermined strings (in other words, 'canned text' *par excellence*) and holes. It is reasonable to assume that, on this occasion, the template as a whole is dependent on some object. The use of the template is therefore historically dependent on that object — likewise the contents of the elements that fill the holes. The template constructs a final text by concatenation, ultimately yielding an object of the same type as the canned elements. Therefore, the template as a whole is not canned, but the predetermined strings are.

The definition also allows canning to be used for any datatype. A syntactic structure containing some leaf nodes annotated with lexical items is canned in just the same way: it contains structures in a configuration of a type which could have been computed by some other process, but those structures are in this case explicitly asserted. The predetermined lexical items are then available for a later process, say inflection, just as would be other lexemes, whose identity would be determined by some other process.

Fig. 4(a) is a general characterization of this situation. Here, b and c are components of a, but of distinct



(a) Generalized canning (b) A concrete instance

Fig. 4. Canning

type, while b, d are of the same type. a, c have histories (indicated by the preceding dashed arrows), while b doesn't. For our first example above, read 'predetermined string' for the type of b, and 'hole' for the type of c. e is then the results of string concatenation (or some representation immediately related to surface presentation). Note that this overall pattern is an instance of a mixed structure as defined above, and the relationship between a and c is that of a partial structure. Fig. 4(b) repeats Fig. 4(a), but substitutes examples of concrete data. In this case, we assume an input such as `request(fax)`. The constructor function is string concatenation.

From this perspective, it is instructive to compare canning with the actions of a simple head-driven sentence realizer, such as that described by [CRZ89]. There, a semantic head is determined by pattern matching a semantic representation against the lexicon. Each act of matching may give rise to subgoals corresponding to argument expressions themselves requiring generation from subparts of the original semantic representation. If no subgoals are generated, the relevant part of the computation is complete. In terms of the picture of generalized canning shown in Fig. 4(a), this amounts to allowing the process of realization which links objects c and d to potentially create new objects of the same type as a.

These examples are sufficient to show that, from an abstract perspective, there is no hard and fast line between "free choice" and template-based approaches. At the template end of the spectrum, transduction will tend to involve more initial objects as there is intentionally less (computed) structure exploited in the generation process, while at the free choice end recursion may be implicated.

It is worth noting here that the approach does make some assumptions about the behaviour of the algorithms manipulating the whiteboard. In order for the generalised canning to function properly, modules need to be 'well-behaved' in the sense that their response to the addition of larger canned structures all in one go must be the same as if the components had been added independently.

7 Future work

There are many aspects of this proposal which we can't discuss in detail here — most certainly in its appropriate technical formulation — and others which we are uncertain of. How, for example, is it best to describe and reason about control? Canning has an impact here, because we may want to use a predetermined structure even when a computed structure might be available (or, indeed, *vice versa*). A second question is the extent to which in practice it will be possible to bring extant generation systems within the scheme proposed here.

8 Conclusions

Our initial goal in developing the whiteboard architecture was to provide a framework that was sufficiently general to subsume the architectures found in practical NLG systems. In order to achieve this goal in a principled way, we had to include formal operations that turned out to support a general approach to canning. With hindsight perhaps this is not so surprising: given that our architecture that is (a) highly modular and (b) not ordered, individual modules need to have the potential to deal with input presented in various states of readiness, and canned structures just represent one extreme on that continuum.

Our characterization of generalized canning in terms of whiteboard configuration permits an abstract view on the manipulation of information in the generation process. From that perspective, the distinction between template-based and "free choice" systems is seen to be essentially one of degree.

References

[CDE+99a] Lynne Cahill, Christy Doran, Roger Evans, Chris Mellish, Daniel Paiva, Mike Reape, Donia Scott, and Neil Tipper. Towards a reference architecture for natural language generation systems. Technical Report ITRI-99-14, University of Brighton, Brighton, March 1999. Also published as HCRC technical report number HCRC/TR-102, University of Edinburgh.

[CDE+99b] Lynne Cahill, Christy Doran, Roger Evans, Chris Mellish, Daniel Paiva, Mike Reape, Donia Scott, and Neil Tipper. In Search of a Reference Architecture for NLG Systems. In *European Workshop on Natural Language Generation*, Toulouse, May 1999.

[CRZ89] Jo Calder, Mike Reape, and Henk Zeevat. An algorithm for generation in unification categorial grammar. In *Proceedings of the 4th Conference of the European Chapter of the Association for Computational Linguistics*, pages 233–240. University of Manchester Institute of Science and Technology, April 10–12 1989.

[Pai98] Daniel S Paiva. Survey of applied natural language generation systems. Technical Report ITRI-98-03, University of Brighton, Brighton, July 1998.

[Ritar] Graeme Ritchie. Completeness conditions for mixed strategy bidirectional parsing. *Computational Linguistics*, to appear.

Time to get real
Current and future requirements for generation in speech and natural language from an industrial perspective

Paul Heisterkamp
DaimlerChrysler AG
Speech Understanding (FT3/AV)
Wilhelm-Runge-Str. 11, D-89081 Ulm, Germany
E-mail: paul.heisterkamp@daimlerchrysler.com

Abstract

In state-of-the-art speech dialogue systems, both the applications behind them and the dialogues themselves become more general and more flexible. The time is near when current template-based generation of system utterances and user information becomes a limiting factor on functionality and, eventually, market success of such systems. They need to be at least augmented with the capability of full-fledged linguistic generation. Linguistic generation systems for speech dialogue, in their turn, have to meet a number of requirements in order to fully serve the purpose of the overall system. We motivate and discuss these requirements, such as real-time processing, ellipses generation, reformulation, choice of mode, time alignment for multi-modal systems etc. We also point to the importance of application creation and maintenance tools.

1. Introduction

In the recent years, the perspective of generation has – as far as we can see it – changed to accept and incorporate 'simple' techniques for creating {spoken|multimedia} language output in generation systems. At this workshop, it really would be carrying coals to Newcastle to argue for the necessity of an integration of both 'free', fully linguistic generation and template-based approaches (cf., e. g. Bateman & Henschel 1999, Calder et al. 1999, van Deemter, Kramer & Theune 1999), as we did a number of years ago (Heisterkamp 1996). So, we need not dwell on the advantages of simplicity. Ease of implementation, ease of maintenance, ease of change and predictability of what's going to be said are mostly recognised benefits of using slot-and-filler templates. But templates require that the structure (or lay-out) of the data they will 'verbalise'[1] is known in the sense that slots are available and filler values are identified, in short, that the 'world' of applications, of things we can talk about, is fully known. Likewise, templates yield only a limited repository of patterns. If someone uses them longer, they can easily become boring, even with a random-choice generator attached. Third, templates are of the 'one-size-fits-all'

[1] The quotation marks indicate that we are referring to multi-modal utterances generally, and use speech and language terms as *pars pro toto*.

kind: their flexibility towards different people in different situations is – to say the least – limited[2]. In short: The benefits as well as the shortcomings of template based generation are obvious.

Currently, for commercial systems, the benefits by far outweigh the shortcomings.[3] The lack of flexibility is not noticed as a limiting factor, as most systems work with one (structurally static) application aimed at the general public. But things are changing. The dynamics of information via the web and the necessity - commercially – to provide custom-tailored services already now require some linguistic, rule-based generation capacity in properly arranging the 'filler values' of slots in templates. With more sophisticated, mixed-initiative dialogue systems on the one hand and information extraction, selection, abstracting, and translation on the other hand coming up fast, the addition of linguistic generation is needed to make these services really useful and economically viable. 'Useful' here means both that the outcome, i. e. the resulting (multi-modal) text serves its purpose for the end-user, as well as that the generation component is easy to adapt, maintain etc. for the service provider(aka 'deployer'[4]). It is from this point of view of 'use-centred' research (cf. Stokes 1997) that we present in this talk some requirements for generation components embedded in overall service-providing systems.

We do not strive for exhaustiveness, nor do we intend to rank these requirements in terms of importance. It is supposed to be a list of research issues and questions like 'Has this been done before, or, does anybody know how to properly do this?', and it would be nice if someone would point at some of the items presented in the talk and say: 'Yes, this is solved, look it up in that-and-that paper'.

In the following, we concentrate on generation for spoken language dialogue systems (or conversational systems: 'synchronous generation' that has to work under the conditions of 'real time' (cf. below).

2. Think in Systems!

The generation component of a dialogue system is crucial in that it is responsible for the only part of the overall system that is 'visible' to the end user[5]. However, it is only useful in so far as it interacts properly with the other components to best exploit their capabilities and to yield a maximum of functionality and naturalness, i. e. ease of use. In particular, this means that the 'what-to-say' part of the generation process is handled by the dialogue management. It determines the next system utterance, the items to be addressed and the type of utterance (question etc.). The dialogue management has to deliver the appropriate surface semantic description. It also should know where the semantic focus of the utterance or particular sentences should be, which items may be elliptified, which items may be anaphorised and perhaps has some idea about 'chunking', grouping of related items. The

[2] For an attempt to enhance the flexibility of templates by endowing them with adaptable prosodic markers, see Hulstijn & van Hessen 1998.

[3] Whether this is due only to the inherent simplicity of templates or to a lack of tool support for other approaches is not the question here.

[4] We have been arguing for some time now to differentiate these two groups what is often still generally call the ‚users' into (end-)users and (system-)deployers, deployers being companies or individuals that offer a service using the technology research and development provide.

[5] Thanks to Peter Poller for this observation.

generator then must negotiate with the dialogue manager about possible and impossible formulations.

On the 'output' side (speech synthesis – concept-to-speech preferably) the generator has to provide not only the linearisation, but also information about prosodic focus, speech-appropriate syntax ('breath groups'), sentence type(s) etc. In case the speech dialogue system has barge-in capability (i.e. the user is allow to interrupt the system as it speaks, and the recogniser can handle this), the generator has to stop the speech output, receive from the synthesiser a time-tag indicating which word was uttered when the interruption occurred, and translate this back into a marker on the respective semantic element and send back the marked and time-tagged order of semantic item to the dialogue manager.[6] The dialogue manager can decide (or at least guess) which semantic item caused the interruption and is the most likely context in which the user utterance can be analysed.[7]

In case of misunderstanding being signalled by the user, the dialogue manager initiates some repair activities. It starts with repairing what the system can repair by itself: its own utterances. The dialogue manager must be able (and allowed to) request a re-formulation of a system utterance (in terms of e. g. explicitness, sentence structure, lexical choice). Now, this is an internal functional argument for the use of 'templates' in 'free' generation. In order to be able to avoid generating the same output, the generator has to store what has been said. This storage feature can be used easily to build up a database of (successful, i. e. un-contradicted) utterances that can then be re-used to speed up the generation process. By the same token, the generator can use this database to improve its own performance by re-ordering its preference of grammar rules, lexical selection, linearisation etc.

3. Real time

In interactive systems, it is essential that the user gets a reaction within the time a human agent would respond. 'Real time' here means the time to the to first sound or image of the system utterance. Within this time, ideally, speech and/or gesture recognition, parsing and dialogue management as well as application system interaction and, not to forget, the actual generation should have taken place. At least in part. Speech output calls for incremental generation.[8] The first difficulty is in doing incremental generation for speech is that speech output is linear: no retracting of backtracking allowed. Now, as it cannot be guaranteed that all of the semantic items that have to be formulated arrive in time to take up their 'correct' position in the utterance, one has to recourse (as humans do) to what is called 'Afterthought syntax', resulting in utterance like 'You want to fly to Rome? From Paris?', where the prepositional phrase 'from Paris?' should have gone *before* the 'to Rome' pp. To implement this type of syntax, we used the ellipsis feature we already had and generated the parts that arrived at linearisation 'too late' by inserting them in the complete utterance and elliptifying

[6] see Poller, Heisterkamp & Stall 1996 and Poller & Heisterkamp 1997 for an implementation of an interface protocol between generator and synthesiser for concept-to-speech synthesis .

[7] This assumes that the interrupting utterance was of error-correcting or similar nature; for an interesting use of purely back-channel interruptions see Iwase & Ward 1998.

[8] In the EFFENDI project, we prototyped an incremental generator for a dialogue system. (EFFizientes Formulieren von DIalogbeiträgen (Efficient formulation of dialogue contributions): Joint project of Daimler-Benz Research and DFKI Saarbrücken 1994-1996; based on ideas published in Heisterkamp 1993; no publicly available papers except Poller, Heisterkamp & Stall 1996 and Poller & Heisterkamp 1997. The bulk of the implementation in EFFENDI and the experimental experience here is due to Peter Poller).

the parts that were already uttered. Thus, we arrived at keeping both the original sentence type and the corresponding prosodic features in intonation and focus. And this is another argument to store and re-use templates also in 'free' generation, as the ellipsis feature in turn is based on consulting a template (record of previous generation).

Even with incremental generation, it may be asking too much to start a system utterance with some 'meaningful' part, especially as connections to application systems (e. g. over the web) can be unpredictable in their time behaviour. This is why it might be useful to gain more time by starting a system utterance with

4. 'Filler' utterances

Apart from being a cheap trick, to start an utterance with some introductory non-semantic word or phrase is very helpful in priming the hearer (user) to raise his or her attention level and to start listening. This is especially true in situations where there may be some distraction. Humans do this all the time. It is a way to ensure the efficiency of the overall interaction in that it helps to be understood. Ditto, in a system that would allow concurrent processing of utterances, it should be possible to generate backchannel utterances (the nods and 'umm's' that signal back attention (not necessarily understanding)).

5. What is efficiency?

'Well, let me get this clear, so this would be a flight to ehemm Paris, leaving from Frankfurt, yes? On Tuesday - so that's the twenty-fourth – next week - at nine o'clock a.m., that's oh-nine-hundred hours , right?' This (not completely made-up) example of a 'system' utterance for a airline schedule inquiry system should illustrate that efficiency of generation in interactive systems is neither the 'shortest' possible formulation nor a fully 'grammatical' one.[9] Rather, it is the formulation that serves best the purposes of being understood, requesting contradiction in case the system did get something wrong (or, as it also happens, the user did not say what he or she 'really' meant) and to give the hearer appropriate time to process the information. In other words, efficiency can be seen as approaching cognitive adequacy. Regrettably, this adequacy is not in the book like grammar rules, it is dynamic in that it is situation and context dependent, and it is hard to measure.[10] Still, it is the overall goal that generation systems should aim at achieving. Another aspect of this adequacy is that it also points at an aspect of interaction that has – to our knowledge – so far not been in the focus of research in language generation, namely the fact that

6. Interaction is a social thing

Currently, spoken dialogue systems are designed to be 'impersonal' in that they do not actively try to achieve the impression of being some specific 'person' (or rather 'persona'). However, people only can model interaction on the basis of human-human communication. This implies a social role of the dialogue partner. So, people do ascribe some sort of personality to the dialogue system they're interacting with, and for whatever reason, we, the

[9] Redundancy is good for communication anyway.

[10] An advantage of interactive systems is that they can exploit user feedback to improve their performance. Backchannel information (Iwase & Ward 1998) could be used in this way; so far, however, we are not aware of any system that would architecturally allow for such a learning feature.

designers, try of counter this by making the systems as neutral as possible.[11] With the advent of 'persona' systems, it is about time to answer such questions as: How do we 'make' an person (impression) through speaking style? And, when and if we con do that (consistently), what use do we put this feature to?[12] Apart from commercial opportunities in the games sector (FinFin's, Talking personalised Tamagotchis and the like), this has immediate implications for the 'communication' (i. e. the advertising and self-presentation) of companies. Does the automated receptionist talk in the way 'corporate identity' requires it, or is it just the ordinary off-the-shelf style? Can I make my product more attractive, differentiate my brand better, by endowing it with a speaking style that suits my target customer group best?

7. Tools: Make it work for everybody

So far, we have been talking about technical requirements, system features and suchlike. Sure, if we have all of these things, we can build really wonderful systems. But we – the developers and techies – are not going to build all the systems that can be created and put into service with our technology. Because we're too few, too long trained and thus too expensive to write a grammar for the umpteenth sports abstract service. If we want to make generation really *useful* (and in the long run, this is the key to commercial success as well) we have to provide mechanisms that allow a broad range of people to take the technology and set up their own system with it, the system that best suits the service, the user and the deployer in terms of functionality, adequacy and (not the least) economy. We have already addressed the topic of self-improvement or learning. Learning currently is mostly done using some sort of statistics. In our view, learning, adapting a system to a new {domain|language[13]|speaking style|multi-modal environment|...} should also be possible by example. Let someone show you how some contents (or semantic representation) should be verbalised, analyse the sentence, and add the style to your rule-base. If you can't do that, use the example as it is whenever possible.

And this is the final argument for the integration of 'free' generation and templates.

[11] Or could this be due to some lack of imagination on our side, we being mostly (some sort of) engineers? Cultural differences seem to matter a great deal here; little wonder that an experiment of 'purely' social interaction (basically, speaking without saying anything, only exchanging social recognition) was conducted in Japan (Suzuki, Ishii & Okada 1998),

[12] A persona speaking in the dialect of an area where people are notorious for being slow could be very helpful in increasing the acceptance for systems with long reaction times.

[13] Multilinguality has been taken for granted so far. It's granted, isn't it?

8. References

Bateman, John; Henschel, Renate (1999): From full generation to 'near-templates' without loosing generality. In this volume.

Calder, Jo; Evans, Roger; Mellish, Chris; Reape, Mike (1999): "Free choice" and templates: how to get both at the same time. In this volume.

Heisterkamp, Paul (1993): Ambiguity and uncertainty in spoken dialogue. In: Proceedings of Eurospeech '93, Berlin.

Heisterkamp, Paul (1996): Natural language analysis and generation. Materials of the course held at the 4th European Summer School on Language and Speech Communication – Dialogue Systems. Budapest, Hungary.

Hulstijn, Joris; van Hessen, Arjan (1998): Utterance generation for transaction dialogues. In: Proceedings of ICSLP '98.

Iwase, Tatsuya; Ward, Nigel (1998): Pacing spoken directions to suit the listener. In: Proceedings of ICSLP '98.

Poller, Peter; Heisterkamp, Paul (1997): A compact representation of prosodically relevant knowledge in a speech dialogue system. In: Proceedings of the Workshop on Concept-to-Speech Generation Systems, July 11, 1997, ACL'97/EACL'97 Joint Conference, Spain.

Poller, Peter; Heisterkamp, Paul; Stall, David (1996): An interface protocol from the speech generator to the speech synthesis module of a dialogue system. In: Bateman, John A. (Ed.): Speech Generation in Multimodal Information Systems and its Practical Applications. Proceedings of the 2nd 'SPEAK!' Workshop (2nd-3rd November 1996 [recte 1995]). St. Augustin: GMD. (GMD-Studien. 302.).

Stokes, Donald E. (1997): Pasteur's Quadrant. Basic Science and Technological Innovation. Washington, DC: Brookings Institution Press.

Suzuki, Noriko; Ishii, Kazuo; Okada, Michio (1998): Organizing Self-Motivated Dialogue with Autonomous Creatures. In: Proceedings of ICSLP '98.

van Deemter, Kees; Kramer, Emiel; Theune, Mariët (1999): Plan-based vs. template-based NLG: a false opposition? In this volume.

TEXTPRO – A Method of Generating Texts from a Formal Language into Natural Languages

Boyd Buchin and Ulf Schmerl

Institut für Theoretische Informatik und Mathematik
Universität der Bundeswehr München, 85577 Neubiberg, Germany
URL: http://infl-www.informatik.unibw-muenchen.de

Abstract. *Attention, the following text has been produced by machine! The corresponding source text is printed at the end of this paper.*

This paper describes the system TEXTPRO which generates texts from the expressions of a formal language into various natural languages. The target languages which we support at present are English, French and German. Since the human user has difficulty understanding the formal language, a more readily comprehensible intermediate language, which until now only exists for English, is intended for each of the target languages.

Résumé. *Attention, le texte suivant a été elaboré à la machine ! Le texte source correspondant paraît à la fin de cet article.*

Cet article décrit le système TEXTPRO qui génère des textes à partir des expressions d'un langage formel dans des langages naturels différents. Les langues cibles que l'on soutient à présent sont l'anglais, le français et l'allemand. Comme l'utilisateur humain a des difficultés à comprendre le langage formel, une langue intermédiaire plus compréhensible, qui jusqu'à présent existe seulement pour l'anglais, est destinée pour chaque langue cible.

Zusammenfassung. *Achtung, der folgende Text ist maschinell erzeugt worden! Der dazugehörige Quelltext ist am Ende dieser Arbeit abgedruckt.*

Diese Arbeit beschreibt das System TEXTPRO, das Texte aus den Ausdrücken einer formalen Sprache in verschiedene natürliche Sprachen generiert. Die Zielsprachen, die wir gegenwärtig unterstützen, sind Englisch, Französisch und Deutsch. Da der menschliche Benutzer Schwierigkeiten hat, die formale Sprache zu verstehen, ist für jede Zielsprache eine besser verständliche Zwischensprache vorgesehen, die bis jetzt nur für Englisch existiert.

1 Introduction

In this paper we describe a method of generating texts in various natural languages from the expressions of a formal language. The formal language codes the information necessary for generating the natural language texts. The target languages under consideration at present are English, French and German. The system TEXTPRO is an implementation of this method.

The intended application of this method is as follows: an author who wishes to publish a text in a language with which he is not familiar or in multiple languages can write the text in the formal language of TEXTPRO. TEXTPRO then takes over the task of formulating the text in the desired target language by means of language specific generators. Compared to machine and human translations, this method has the obvious advantage that there is no need for an analysis of the natural language source text. This analysis is a problem which has, to date, been insufficiently resolved.

To facilitate the use of the formal language, we intend to construct an intermediate language for each of the target languages. The intermediate languages should be constructed in such a way that they are more user-friendly than the actual formal language and can be easily "compiled" into the formal language by computer. We plan to achieve this by using disambiguated fragments of speech and thus allow the user to work with a fragment of his mother tongue. So far only the intermediate language for English is available.

2 Similar Systems

Our intention has been to design TEXTPRO in such a way as to cover a sufficiently large and general fragment of speech of the target languages. The fragment should be sufficient for formulating technical and mathematical texts. This rules out using pre-formulated text-blocks as they can only cover specific fragments of speech and thus their use is limited to special applications. We therefore based our system on a formal language. In objective, our approach shows a fair degree of similarity to those of KPML, FUF, the COGENTEX-family and, possibly, UNL. They too are based on formal languages, which differ – in some cases considerably – from ours (Paiva 1998, Senta and Uchida 1998). The differences prevent a mapping of our formal language into one of the others, and thus the re-use of an existing generator.

3 TEXTPRO: Theory and Implementation

The theoretical basis of the TEXTPRO-project is the development of the formal language together with its versions as intermediate languages. Its implementation, the TEXTPRO-system, is made up of compilers, which translate texts from the intermediate languages into the actual formal language, and generators, which translate texts from the formal language into the target languages.

3.1 The Formal Language

As we already mentioned above, the formal language has the function of coding the information necessary for generating the natural language texts. We decided on a sentence-oriented representation of this information, as the structure of sentences can more easily be precisely framed by grammatical rules than the structure of texts. We let ourselves be guided by heuristic principles in taking advantage of syntactic agreement between the three target languages, i. e. by taking over corresponding syntactic structures into the formal language.

We circumvent differences in the syntactic structure of the three languages by using instead more profound semantic descriptions. This surmounts the differences, without trying to find absolute semantic descriptions and thus attempting to construct a "universal interlingua". An example of a difference which has to be resolved in this way is the negation in English and German. If one takes the sentence

Peter does *not* see Mary. (1)

one is tempted to formalise it as

$$\neg\,(\textit{see Peter Mary}) \qquad (2)$$

which can be generated into German as:

Peter sieht Maria *nicht*. (3)

But if one tries to formalise the very similar sentence

Peter does *not* see many guests. (4)

in the same way as we formalised sentence (1), the result is

$$\neg\,(\textit{see Peter}\ (\textit{many guest})) \tag{5}$$

which corresponds to the German:

Peter sieht viele Gäste *nicht*. (6)

However, a correct translation of sentence (4) into German would have been:

Peter sieht *nicht viele* Gäste. (7)

The reason why the naive approach fails in the second example is that the negation semantically refers to the quantifier *many*. This can be made visible by putting sentence (4) into the passive:

Not many guests are seen by Peter. (8)

One way to overcome this difference in syntax is to use generalised quantifiers like Barwise and Cooper (1981) propose. In our example it is the generalised quantifier $\neg$ *many* which allows a sufficient formalisation of sentence (4):

$$\textit{see Peter}\ (\neg\,\textit{many guest}) \tag{9}$$

Negation is not the only phenomenon where a naive approach fails. Other differences in syntax can be found in articles, modal auxiliaries, voice, tense and aspect.

Even so the three target languages mentioned above show a large measure of agreement in their sentence structure and other syntactic structures, e. g. relative clauses, *that*-, infinitive- and *wh*-clauses, constructions of noun phrases, prepositional phrases and adverbials, which can be used for the formal language. In this respect, our approach differs considerably from that of UNL, where the aim is to generate a text into dozens of highly different natural languages, which in all probability possess few syntactic correspondences.

As the given space does not allow us to present the whole formal language, we will focus on how sentences are formalised in TEXTPRO. TEXTPRO distinguishes between two types of sentences: the simple and the composed sentence. Simple sentences are sentences which do not have other sentences as direct constituents. Composed sentences, on the other hand, are constructed from other sentences joined together by a connector such as *and*, *or*, *but*.

Simple sentences are either mathematical formulae (as in *if* $x = 0$ *then* ...) or consist of a predicate and a list of terms. Each predicate has its own valence, which states the number of terms the predicate requires. These mandatory terms correspond to subject and objects of the sentence. All other terms are called free and represent adverbials.

Usually predicates take noun phrases as mandatory terms, but some require *that*-, infinitive-, *wh*-clauses or other types of expressions at certain positions (*I know that he will come*).

Thus the valences of predicates not only define the number of required mandatory terms but also their type. Finally some predicates permit terms of different types in certain positions (e. g. *he proved the theorem*; *he proved that 2 is a prime number*), so that some valences must furthermore be oligomorphic. A simple sentence thus has the structure:

$$p^{\tau}\, t_1^{\tau_1} \ldots t_n^{\tau_n}\, f_1^{\alpha_1} \ldots f_m^{\alpha_m}, \text{ if } \langle \tau_1, \ldots, \tau_n \rangle \in \tau$$

τ is the valence of the predicate p, $t_1^{\tau_1}$, ..., $t_n^{\tau_n}$ are the mandatory terms, whose types are τ_1, ..., τ_n, and $f_1^{\alpha_1}$, ..., $f_m^{\alpha_m}$ are free terms, whose types are α_1, ..., α_m. Of course predicates, mandatory terms and free terms are also complex structures. For example, a predicate is divided into a kernel and modifiers. The kernel is either a verb or an adjective and the modifiers make it possible to express modality, tense, voice and aspect, among other things. The structure of the terms is even richer. For a detailed description we refer to Buchin (1999).

3.2 The Implementation of the TEXTPRO-System

At present the TEXTPRO-system consists of four main components: a compiler, which translates sentences of the English intermediate language into the actual formal language, and three generators for the target languages supported at present, namely English, French and German. In its final form, TEXTPRO will consist of a compiler and a generator for each the n target languages – resulting in a total of $2n$ components.

We defined the compiler for the intermediate language for English in the form of a context-free grammar. It consist of over 260 production rules and distinguishes close to 80 different types of terminals. The parser-generator HAPPY converts it into program code. This approach has the advantage that the definition of the compiler is compact and can be checked for ambiguities automatically.

The generators of TEXTPRO are written in the functional programming language HASKELL. We chose this language because of its strength in list processing: lists play an important role in natural language generation and in HASKELL even complex list transformations can be expressed with ease. In total, the system consists of about half a megabyte of code. This does not include the dictionaries. They are realised as text files and currently contain about 1000 entries per language. Both the compilers and generators make use of them.

We will not describe the components in further detail, but instead present the (ASCII)-text, from which TEXTPRO generated the abstracts at the beginning of this paper. It gives an impression of the task TEXTPRO has to complete:

|**Abs Abstract**|. %%\emph{%% Attention, the following text H B produced$_2$ by machine! The corresponding$_2$ source text is ‹printed› at {the end of this paper$_2$}. %%}%%

This paper$_2$ describes {the system %\textsc{TextPro}%} which λt(t generates texts from {typ expressions of a formal language} into various natural languages). The target languages which λt(we{} support t at present) are [English, French and German]. Since (the human user has difficulty ε(understanding the formal language)) for each of the target languages a more readily comprehensible intermediate language which $\lambda' t$(until now, t only exists for English) is ‹intended›.

References

Barwise, J. and Cooper, R. (1981). Generalized quantifiers and natural language. In *Linguistics and Philosophy*, volume 4, pages 159 – 219. Reidel.

Bateman, J. (1997). Some apparently disjoint aims and requirements for grammar development environments: the case of natural language generation. In *Workshop on Computational Environments for Grammar Development and Linguistic Engineering (ACL-EACL '97)*, Madrid.

Buchin, B. (1999). *Die Generierung natürlicher Sprache aus einer formalen Repräsentation.* PhD thesis, Universität der Bundeswehr München, CIS-Bericht 99-122, Universität München.

Duroux, P., Preller, A., and Schmerl, U. (1995). Natural language text generation from a formal language. Université Paul Valery, Montpellier.

Elhadad, M. and Robin, J. (1999). *SURGE: A Comprehensive Plug-in Syntactic Realization Component for Text Generation.* Ben Gurion University.

Kittredge, R. and Lavoie, B. (1998). Meteocogent: A knowledge-based tool for generating weather forecast texts. In *Proceedings of the American Meteorological Society AI Conference (AMS '98)*, Phoenix, Arizona.

Paiva, D. S. (1998). *A Survey of Applied Natural Language Generation Systems.* Information Technology Research Institute (ITRI), University of Brighton.

Senta, T. D. and Uchida, H., editors (1998). *Universal Networking Language - UNL.* UNU/IAS Tokio.

The VOLVEX Handbook - A General Validation Tool by Natural Language Generation for the STEP/EXPRESS Standard

Hercules Dalianis
Department of Computer and Systems Sciences (DSV)
The Royal Institute of Technology (KTH) and Stockholm University
Electrum 230, S-164 40 Kista, SWEDEN,
E-mail:hercules@dsv.su.se

Abstract

STEP Application Protocols (APs) are often very large and complicated general descriptions of different domains mainly within the manufacturing industry. STEP AP's are expressed in the EXPRESS language which is a static modeling language of Entity-Relationship type. Users of STEP AP have often large problems to understand the whole AP and therefore they need a tool which helps them to validate the STEP AP. The tool we are proposing is a natural language English paraphraser of the STEP AP. In this handbook we are demonstrating how to automatically build a AP domain lexicon from one STEP AP and with this lexicon automatically translate one arbitrary STEP AP EXPRESS file into a Prolog based format to be used by a Natural Language Generator called ASTROGEN. The translation from EXPRESS format to EXPRESS Prolog format and the construction of the domain lexicon is carried out by a set of Perl programs.

1. Introduction

To make STEP AP's and formal language descriptions understandable for "naive" users one needs to paraphrase them to natural languages by using natural language generation (NLG) techniques. Two serious obstacles prevent the use of NLG systems for large collections. First, the limited domain lexicon and the effort to extend it to construct the domain lexicon is time-consuming and costly. Second, the generated text may seem too "computer-made" and therefore boring. For example, the text below, (see Figure 1), is automatically generated from the STEP Application Protocol 214 (AP214), the automotive design application protocol [Al-Timimi and MacKrell, 1996], using neither text nor sentence planning, nor canned or example text.

To produce even poor text like this (Figure 1) for a new domain, the user is required to build up a lexicon of new terms, this can be a time-consuming task. To improve the quality of such machine generated text, the user may re-use some fragments of the STEP AP concept definitions verbatim (after all, they *were* produced by humans!) and also make use of the sentence planning abilities of the ASTROGEN Natural Language Generator [Dalianis, 1997] to generate texts that are nice to read.

```
?- question(project & project_relationship).
a project is an entity and
a project has an undefined_object id and
a project has a string_select name and
a project has a string_select description and
a project has a date_time actual_start_date and
a project has a date_time actual_end_date and
a project has an event_or_date_select planned_start_date
and
a project has a product_class affected_product_class and
a project has an activity work_program and
a project has a period_or_date_select planned_end_date
and
a project_relationship is an entity and
a project_relationship has a project related and
a project_relationship has a project relating and
a project_relationship has an undefined_object
relation_type and
a project_relationship has a string_select description.
yes
```

Figure 1. Automatically generated natural language text from AP214.

2. Background

STEP stands for STandard for the Exchange of Product model data, and is an ISO 10303 standard [Al-Timimi and MacKrell, 1996]. STEP has been developed by industry for the exchange of product model data between different platforms, as e.g. CAD/CAM platforms.

The STEP standard contains Application Protocols (APs) that are standardized schemata within each domain, expressing the standardized concepts. Each AP consists of two files: the formal concept definitions and the associated text definitions. APs exist in several domains, including automotive manufacturing, ship building, electrotechnical plants, and process industry. The APs are expressed in the data modeling language EXPRESS [Schenk and Wilson, 1994] EXPRESS is a static modeling language of Entity-Relationship type.

3. Previous Research

Several attempts have previously been made to automate the lexicon acquisition for natural language interfaces, e.g., TEAM [Grosz et al., 1987] and CLE [Alshawi, 1992]. In both TEAM and CLE, the domain users had to answer a set of question for each lexical entry and hence identify the entry. In TELL [Knight et al., 1989], another approach was made, a set of heuristics were used together with the CYC Ontology [Lenat & Guha, 1988] to identify the lexical entries. The requirements engineering community [Chen, 1983] has been identifying the relations between various entities in conceptual models and word categories.

Our view is that there is no user in a real industrial setting who is really interested in answering thousands of questions to acquire a lexicon. This argument is also reflected in [Reiter and Mellish, 1993], where the authors say that to make NLG more useful and practical, one needs to make the customization process fast and efficient. Discussing the costs and benefits for NLG, they also argue that the only way to make NLG techniques competitive is to use its advantages (flexibility in the produced texts) without its disadvantages (costly lexical acquisition and knowledge base building). Therefore in this paper our approach is based on the assumption that the STEP schemata, per se, contain all necessary linguistic information to create a domain lexicon. The approach can be seen as something between TELL [Knight et al., 1989] and [Chen, 1983].

We also argue for the use of hybrid systems, a term coined by Reiter [Reiter, 1995]. These are systems that use a combination of techniques from traditional NLG systems and canned texts. Hybrid systems have turned out to be very practical since canned texts always are available somewhere and just need to be combined with real generated text. [Mittal and Paris, 1993] present an example of a hybrid system using a combination of natural language generated text and examples to make the explanation more user-friendly. A similar approach was made in [Dalianis et al., 1997] after user studies indicating that schema information not available in the schema is used for manual paraphrasing of schemata by domain experts.

4. The Acquisition of the Lexicon and Canned Text Definitions

To extract the lexicon and the canned text from the STEP files and the definition files respectively we use a set of Perl programs. Perl , [Wall et al., 1996], is excellent for string processing. For the STEP domain in general we have an express base-lexicon [Dalianis et al., 1997] that contains all lexical terms used in the EXPRESS language. What we need to construct is a domain lexicon for each domain or AP, as well as the set of canned texts expressing definitions and examples (see Figure 1).

Altogether from the STEP AP214, the automotive design application protocol, we created 1551 lexical objects (1291 nouns and 260 adjectives), 492 canned definition texts, and 106 canned example texts.

Extraction of nouns and adjectives

The lexical extraction program scans the EXPRESS files for EXPRESS entities and attributes which are extracted as nouns and adjectives respectively (according to [Chen, 1983]) and saved in a lexicon file with extension as a Prolog DCG-clause [Clocksin and Mellish, 1984] reflecting the lexical items.

Extraction of canned definition text

The canned text extraction program scans the EXPRESS definition files for definitions corresponding to a specific entity, which are saved to a file, with extension as a Prolog fact, together with the specific entity as a key.

The text definition file contains textual information in natural language (NL) form of each entity. We extract only the first sentence of each text description since we have the impression that this gives a fair overview description of the entity. In many cases the full text description is cumbersome.

Extraction of canned example text

The canned text extraction program scans the EXPRESS definition files for examples corresponding to a specific entity, which are saved to a file, with extension, as a Prolog fact together with the specific entity as a key.

5. The ASTROGEN (Natural Language) Generator

The ASTROGEN generator [Dalianis, 1997] is written in Prolog. ASTROGEN has its main strength in its aggregation rules, [Dalianis and Hovy, 1996], that remove redundant portions of a text without changing the content. In the ASTROGEN documentation one can read about the use of the ASTROGEN generator and also download the whole generator.

ASTROGEN takes as input a set of content-selected f-structures (an internal representation) and performs first sentence planning: it applies the aggregation rules to the f-structures, carries out pronominalization on the aggregated result, then creates a coherent discourse structure of the f-structures, and second with this as input, the surface generator then generates the syntactic surface structure and the lexical objects. Finally, the sentence transformer performs the post-processing of

the text.
We customized the ASTROGEN generator for the EXPRESS language with an express base lexicon, defining specific EXPRESS reserved words.

To make use of Prolog's extremely efficient matching capability we translated the whole STEP AP (EXPRESS) file to Prolog syntax. This made it possibly to ask questions about the STEP AP and hence make content determination from the abundant knowledge base of the STEP AP.

Sentence transformation
As a final step we perform a set of sentence transformations to blend the generated output, which consists of both generated sentences and fragments of canned text, together. The sentence transformation rules are applied on the final output NL string. Two main heuristics are carried out. First, each first letter of a sentence is capitalized so the text will look more natural when displayed together with the canned definitions and example texts. Second, the aggregated (coordinated and elipted) [Dalianis and Hovy 1996] text is post-processed by replacing consecutive *and*s with commas, except for the final *and.* The canned texts (and example texts) are reproduced just as they are stored in the definition files; no sentence transformation is carried out on them.

6. Generating Concept Descriptions

When all the above mentioned preparations were complete, we generated concept descriptions from the STEP AP 214 in NL, using ASTROGEN. Two examples are shown in Figure 2. Each concept description contains sentences describing the supertype, subclasses and attributes of the concept (produced by ASTROGEN) and the canned definitions and also when available the canned example. As can be seen from Figure 4, the generated and canned texts fit together nicely to give a fairly coherent result.We produced descriptions for 501 concepts from the STEP AP214.

```
?- question(fillet).
A constant_radius_fillet is a subtype of a fillet.
A fillet is an entity.
It is a subtype of a transition_feature. (Pron.)
A Fillet is a concave circular arc transition between two intersecting Face (see 4.2.167) objects
without any constraints concerning changes of the radius along the Fillet. (Canned text)
yes

?- question(project & project_relationship).
A project and a project_relationship are entities. (Agg.)
They have a string_select description. (Pron.)
A project has a date_time actual_end_date.
It has a date_time actual_start_date. (Pron.)
It has a product_class affected_product_class. "
It has an undefined_object id. "
It has a string_select name. "
It has a period_or_date_select planned_end_date."
It has an event_or_date_select planned_start_date."
It has an activity work_program. "                    (cont')
```

```
A project_relationship has a project related.
It has a project relating. (Pron.)
It has an undefined_object relation_type. "
A Project is a unique process with a time limit, with a defined goal, with a defined budget, and with
defined resources. (Canned text)
A Project_relationship is a relationship between two Project (see 4.2.356) objects. (Example text)
EXAMPLE 174 -- For the development of a new car, a project is set up that is responsible for the
development decisions as well as for the accounting of the costs.
yes
```

Figure 2. The output from of the ASTROGEN generator describing STEP AP214, Italicized comments indicate processing steps.

7. Conclusions and future directions

In this paper we describe a fast and efficient method to build a natural language generation system for a real industrial setting. This work has been carried out by building the lexicon and adapting the database to Prolog and to ASTROGEN automatically from STEP APs.

Future work will be to adapt this generation technique to a similar domain namely the UML Unified Modeling Language which is a new standard in software engineering. UML is similar to EXPRESS but has dynamics.

Our plans are also to integrate the results of this paper with the VINST tool [Dalianis, 1998], in order to provide the user with extracts of STEP Schemata translated to NL. Future basic research will be to elaborate on the extraction of nouns from entities and adjectives from attributes and to extend the sentence and text planner.

Acknowledgements

Great thanks to Dr Eduard Hovy and USC/Information Sciences Institute for interesting and fun discussions around automatic integrations and to Dr.Chin-Yew Lin and Uli Germann also at USC/Information Sciences Institute for their help in the art of programming Perl.

I would also like to thank my sponsors Volvo Research Foundation, Volvo Educational Foundation and Dr Pehr G Gyllenhammar Research Foundation for their support to the VOLVEX project- **V**alidation **O**f Specifications by Natural **L**anguage Generation for **VO**LVO expressed in STEP/**EX**PRESS.

8. References

Alshawi, H. (Ed.) 1992. The Core Language Engine, *MIT Press.*

Al-Timimi, K. and J. MacKrell. 1996. STEP Towards Open Systems. *STEP Fundamentals & Business Benefits*, CIMdata.

Chen, P. P-S. 1983. English Sentence Structure and Entity Relationship Diagrams, *Information Sciences 29(2)*, pp. 127-149.

Clocksin, W.F. and C.S. Mellish. 1984, *Programming in Prolog.* Springer Verlag.

Dalianis, H. and E. Hovy. 1996. Aggregation in Natural Language Generation. In Adorni, G. & Zock, M. (Eds.), *Trends in Natural Language Generation: an Artificial Intelligence Perspective*, EWNLG'93, Fourth European Workshop, Lecture Notes in Artificial Intelligence, No. 1036, Springer Verlag. pp. 88-105, .

Dalianis, H. 1997. ASTROGEN-Aggregated deep and Surface naTuRal language GENerator, http://www.dsv.su.se/~hercules/ASTROGEN/ASTROGEN.html

Dalianis, H, P. Johannesson and A. Hedman. 1997. Validation of STEP/EXPRESS Specifications by Automatic Natural Language Generation. In *Proceedings of RANLP'97: Recent Advances in Natural Language Processing*, pp. 264-269. Tzigov Chark, Bulgaria, September 11-13, 1997.

Dalianis, H. 1998. The VINST Approach:Validating and Integrating STEP AP Schemata Using a Semi Automatic Tool. In N. MŒrtensson et al (Eds). Changing the Ways We WorkShaping the ICT solutions for the Next Century, IOS-Press, 1998, pp. 211220. *Proceedings of the Conference on Integration in Manufacturing (IiM-98)*. Gothenburg, Sweden, October 68, 1998.

Grosz, B.J., D.E. Appelt, P.A. Martin, and C.N.Pereira. 1987. Team: An Experiment in the Design of Transportable Natural-Language Interfaces, *J. Artificical Intelligence 32(2)* pp. 173-243.

Knight, K., E. Rich and D. Wroblewski. 1989. Integrating Language Acquistion and Knowledge Acquisition. In the *Proceedings of First International Workshop on Lexical Acquistion JCAI-89.*

Lenat, D. and R.V. Guha. 1988. The world according to CYC. Tech Report ACA-AI-300-88 MCC (Microelectronics and Computer Technology).

Mittal, V. and C. Paris. Automatic Documentation Generation: The Interaction of Text and Examples. *Proceedings of 13th International Joint Conference on Artifical Intelligence, IJCAI-93,* pp. 1158-1163.

Reiter, E. and C. Mellish. 1993. Optimizing the Costs and Benefits of Natural language Generation. Proceedings of *13th International Joint Conference on Artifical Intelligence, IJCAI-93*, pp. 1164-1169.

Reiter, E. 1995. NLG vs. Template. In *Proceedings of the Fifth European Workshop on Natural Language Generation*, Leiden, The Netherlands. Schenk, D. and P. Wilson 1994. *Information Modeling the Express Way,* Oxford University Press.

Wall, L., T. Christensen, and R.L. Schwartz. 1996. *Programming Perl.* O'Reilly & AssociatesInc.

A Multilingual Text Generator for Real-Time WEB-Communication

Annette Fritsch, Eric Cousin, Philippe Tanguy

July 7, 1999

ENST Bretagne
Technopôle Brest-Iroise, BP832
29285 BREST Cedex
Annette.Fritsch@enst-bretagne.fr

1 Introduction

Multilingual natural language generation as a tool of interaction on the web is more than ever an issue of highest interest and will be of major concern in various web-situation, such as reservations, virtual shopping, talks/chats or virtual worlds, and information seeking. As communication on the Internet is about to become a means of interaction for everyone, both commercially and as a way to spend leisure time, the use of the mother tongue will indeed become more important, as it is quicker, more efficient and personalised. In that, we agree with Boitet [1], who highlights that "to take over English as the unique language of communication is not cost effective. There is a strong desire to use ones own language, while of course trying to learn a few others for personal communication and cultural enrichment".

In order to enhance multilingual interaction and communication, we developed a context-dependent multilingual application system on a template based approach. This sort of "bridge the gap"-tool between two users unable to communicate in the same language allows real-time synchronic written communication within a chosen context.

This approach is not research within large-scale linguistic resources, such as KPML [6] for instance. Work that has been carried out in this area proved that the context-free approach and its linguistic constraints are far too complex to provide successful translation for everyone in every context on a low-cost level [2], [4], [5]. Our goal is rather to provide a very practical, pragmatic approach towards cross-cultural comprehension, communication and interaction. We limit ourselves to reachable goals by systematically focusing on different contexts in order to achieve successful communication, i.e. syntactically, semantically, and pragmatically correct translations. We see our work as a part of HCI in that "better human-computer interaction strategies have to be developed, as multilingual language translation becomes a tool to broker an understanding between two humans rather than a black box that tries to translate every utterance" [7].

2 Linguistic requirements

The underlying idea was that the linguistic structures we would need for our purposes were likely to be recurrent question-answer speech-acts, comparable to adjacency-pairs in telephone calls [3]. We consider that these adjacency-pairs are recurrent patterns in commercial contexts such as hotel reservations, department store-call centres, and ticket booking, and that we can adapt them to online hotel reservation services or online ticket booking. The same account for the domain of virtual games.

We believe that for our restricted purpose, focusing on question answer speech-acts comes closest to what the online situation requires.

We focus on a combination of a question-answer corpus with a filler-words corpus, which leads to a fairly large number of questions in different variations. Furthermore, both corpora can be specifically shaped according to the context they have to operate in. The corpus may be established by analysing data from real-life conversation such as data from call-centres or any other real-life environment that might be stored in the corpora. This context-dependent framework allows us to rule out the semantic and pragmatic ambiguities, which are inevitably met when working within a context- independent approach.

3 Usability

Since we want to avoid slow and incorrect, thus frustrating communication, we opt for a keyword-based system. The user enters keywords and the system generates a range of sentences containing these keywords. Entering the keyword room for instance, the system might suggest "Do you want to book a room?" in the hotel context. The user selects the sentence that suits his purposes best. The system is then able to generate instantly a correct translation of the chosen sentence in any of the known languages. By choosing this method, we want to avoid unsuccessful translation due to entering false spelling, as well as slowness in translation speed. At the moment, our system processes English, French, Spanish and German. The keyword approach works quite well as long as the corpora are not too large. A user who gets lost and who has too choose between an enormous amount of sentences offered by the system cannot be in our interest either. Thus, we need to find a compromise between quantity and quality, i.e. a corpus which is sufficient without confusing the user. Nevertheless, this is rather a question of stocking sentences and overall organization than a problem of large corpora.

4 Test-bed

We consider that games in general, as a sort of prototypical interaction amongst people, (guessing games in particular), are a good means to test the feasibility of a combination of sentence-structure corpus with a filler word corpus. Games combine the question-answer-structures of commercial conversations which we are likely to need and the fun-factor side which we do not want to neglect, since we want to test the users acceptance towards the keyword approach. The multilingual tool has been tested with a ten questions guessing game which we adapted specially for this purpose. In order to enhance the fun-factor, we created the possibility of choosing between the different style levels of communication: normal, colloquial and sophisticated style. This will certainly be of somewhat different importance within a commercial context, but was very much appreciated by our test players within the game context. The players were able to interact with the given set of available recurrent linguistic patterns at their disposal, and managed indeed to have a successful multilingual interaction within the given context. Nevertheless, both corpora have to be enlarged in order to conduct tests on a larger scale.

5 The Multilingual Tool

The tool is a multilingual text generator based on two corpora, the sentence-structure corpus and the filler-word corpus, which we call, in the computerised version, Sentence-pattern manager and Filler-word manager. The overall architecture of the tool is based on a system of index matching.

Thus, words from the Filler-word manager are inserted by the Sentence-pattern manager according to the index-compatibilities. The following examples are extracts from one of the test games, where the players had to guess an animal. For this purpose, both corpora were adapted to the animal context. Since the formal structures and the overall architecture of the system is running, we are positive that adaptation to a different context is only a question of a few weeks, the case given that the real-life corpus has already been established.

5.1 Sentence-pattern manager

Figure 1 shows the sentence-patterns with their markers. In the examples given, the markers V1, V2 and N2 indicate whether the sentence pattern requires a transitive infinitive, a transitive present or a plural noun.

The indexing system receives the information needed by using the semantic key (S), indicating that the different sentences in the different languages do correspond. One of the advantages of the system is inherent in the fact that it does not work on a language-pair basis. It is constructed around the markers. Therefore, any other language can be added, in case we meet the morpho-syntactic requirements.

Since we place ourselves in the game-context, the player can choose between different style levels, managed by the style-level key (R). The style-level approach might be altered when dealing with other contexts. While games certainly demand a more emphatic conversational style, other contexts, such as hotel booking, focuses on the purely informative level.

S	R	Sentences
33	0	est-ce qu'il V2 des N2 ?
33	1	pourriez vous m'aider et me dire s'il V2 des N2 ?
33	2	ça V2 des N2 ce truc là ?
33	0	does it V1 N2 ?
33	1	be so kind as to tell me, does it V1 N2 ?
33	2	it V2 N2, right ?
33	0	V2 N2 ?
33	1	podemos pensar que el animal V2 N2 ?
33	2	aquello V2 N2 ?
33	0	V2 es N2 ?
33	1	Muss ich davon ausgehen, dass es der Lage ist N2 zu V1 ?
33	2	Hier, sachma, tuts N2 V1 ?

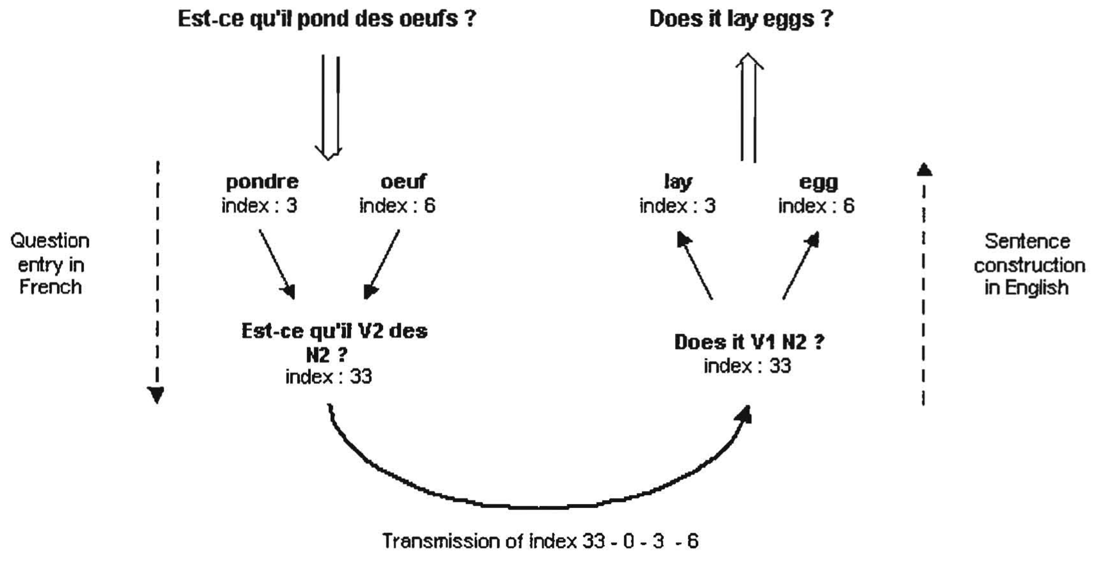
5.2 Filler-word manager

Figure 2 shows the filler word manager. Every word is given with its derived forms and classified by its grammatical type (T) as well as its inter- language properties (I2). The inter-language properties such as gender, number, and conjugation correlate with the possibilities of insertion in the sentence pattern manager.

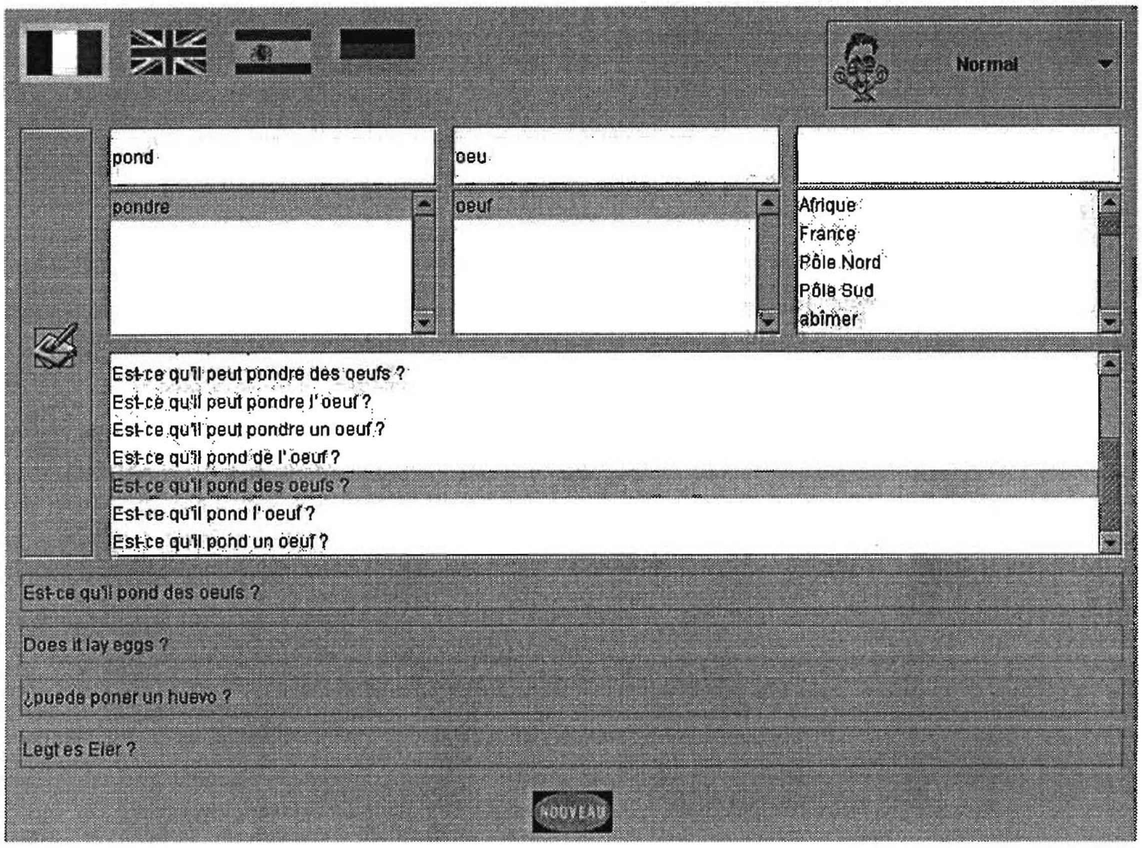
		French		English		Spanish	
I1	T	Forms	I2				
1	V	chanter/chante	1	sing/sings	1	cantar/canta	1
2	V	chasser/chasse	0	hunt/hunts	0	cazar/caza	0
3	V	pondre/pond	2	lay/lays	2	poner/pone	2
1	V	chanter/chante	1	sing/sings	1	canntar/canta	1
4	N	léopard/léopards	5	leopard/leopards	4	leopardo/leopardos	5
5	N	lièvre/lièvres	5	hare/hares	4	lievre/lievres	5
6	N	oeuf/oeufs	7	egg/eggs	5	huevo/huevos	10
7	ADJ	joli/jolie/jolis/jolies	10	attractive	7	guapo/guapa/ guapos/guapas	10
8	ADJ	gris/grise/gris/ grises	10	grey	6	gris/gris/grises/ grises	9

6 General architecture of communication

Figure 3 gives a graphical description of the principle of "translation". The user chooses the sentences he or she wants to be translated. The system generates the sentence by applying the semantic key index (33) the register-level index (0) and the index of the words to be inserted (3 and 6). Then the system tests the compatibility of the information given.


Est-ce qu'il pond des oeufs ?
Does it lay eggs ?
pondre index : 3
oeuf index : 6
lay index : 3
egg index : 6
Question entry in French
Sentence construction in English
Est-ce qu'il V2 des N2 ? index : 33
Does it V1 N2 ? index : 33
Transmission of index 33 - 0 - 3 - 6

Communication takes place via an applet, which pops up on the interface if the player wishes to use it.



Est-ce qu'il peut pondre des oeufs ?
Est-ce qu'il peut pondre l'oeuf ?
Est-ce qu'il peut pondre un oeuf ?
Est-ce qu'il pond de l'oeuf ?
Est-ce qu'il pond des oeufs ?
Est-ce qu'il pond l'oeuf ?
Est-ce qu'il pond un oeuf ?

Est-ce qu'il pond des oeufs ?

Does it lay eggs ?

¿puede poner un huevo ?

Legt es Eier ?

7 Limitations and further developments

So far, we have tested and validated the tool in the context of animal guessing. Having achieved successful communication between players within this context, we believe that an application in i.e. enigmatic games in virtual worlds may lead to coherent multilingual communication between participants. Furthermore, translation will be quick, which is an advantage not to be underestimated, given the slowness of current systems and page loading. Thus, it is a real nuisance for the user to have to go back to, i.e. Systran, enter the sentence, wait for the answer, and go back to the site, still not knowing if the sentence will be comprehensible to his interlocutor.

Other applications of the system may involve all information-seeking and commercial contexts, i.e. virtual call-centres, where an immediate response is necessary, instead of delayed response via e-mail. From the technical point of view, the applet will be inserted in the HTML page. Nevertheless, linguistic preliminary work has to include an analysis and definition of the speech-acts involved and needed for such purposes. In the following, the two corpora have to be adapted to the given context and filled with the vocabulary and the sentence structures involved. It is obvious that such a system is of very limited use as soon as the context is not closely defined.

Presently, we only operate on the basis of Indo-European languages, whose morpho-syntactic structures are quite easy to handle. Although we are positive that basically all Indo-European languages can be added to the system, we will be in trouble as far as agglutinating languages such as Finnish or Hungarian are concerned : the system is not able to produce alternation at the stem. Future research will have to include trying to find a means of adding an alternation at the stem-component, at least for the regular cases.

8 Conclusion

We do believe that work such as ours can help pave the way for developments of user-centred realistic translation services that can help to enhance and simplify multilingual human-computer-human interaction. This does not mean, that we do not see the necessity of research within the field of deep techniques for NLP, on the contrary. Nevertheless, we believe that the two approaches can exists side by side, and that shallow techniques might be useful as an ersatz, allowing more people access to a wider range of services and cross-cultural communication, as long as large scale linguistic resources are still on their way.

9 Aknowledgments

This work was partly supported by European fundings (Amusement project - ESPRIT LTR project 25197).

Special thanks to Josette Jouas, Jose Manuel Abreu, Daniel Bourget and the students who were involved in the work reported here.

References

1. Boitet, C. (1996) (Human-aided) machine translation : a better furture ? In Survey of the State of the Art in Human Language Technology : http://cslu.cse.ogi.edu/HLTsurvey/ch8node5.html#SECTION83

2. Chandioux, J Les promesses de la traduction automatique http://www.riofil.org/oqil/tao/tradauto.htm (1999)

3.Ervin-Tripp, S. (1972) On sociolinguistic rules: alternation and co-occurrence In J.J Gumperz & D. Hymes (eds.) Directions in sociolinguistics. New York: Holt, Rinehart & Winston, 213-50.

4.Language Partners International (1999) An Introduction to Computer Aided Translation (CAT) http://www.languagepartners.com/reference-center/catintro.htm

5. Lauckner, C Traducteurs automatisés - Etude bibliographique, ENST Bretagne (1998)

6. Stirling university The KPML multilingual natural language generation system, development environment and tools (1999) http://www.stir.ac.uk/english/communication/Computational-tools/kpml.html

7. Waibel, A. (1996) Multilingual Speech Processing In Survey of the State of the Art in Human Language Technology: http://cslu.cse.ogi.edu/HLTsurvey/ch8node8.html#SECTION86

Templates for Wearables in Context

Sabine Geldof

Vrije Universiteit Brussel
Artificial Intelligence Laboratory
Pleinlaan 2, B-1050 Brussels, Belgium
sabine@arti.vub.ac.be
http://arti.vub.ac.be

1 Introduction

In the COMRIS[1] project we are developing natural language generation (NLG) technology for output on a wearable device: a user, moving freely in the area of a conference receives from her COMRIS parrot (spoken) advice (e.g. on events and encounters not to miss, reminders about commitments and proposals for meetings). This setting requires NLG technology that is fast and context sensitive: otherwise the user will be bored and turn off her device. In contrast to other NLG projects for wearable devices, e.g., HIPS [1], the COMRIS parrot is supported by a mixed reality set-up. A multitude of agents, defending users' interests and aware of their physical context, are continuously interacting in the virtual space, eager to pursue the specific (user) interest they stand for. They try to push the results of their activities to the physical space in the form of short, ad-rem messages to the user. A mechanism of competition for attention [2] ensures that only the relevant messages are passed to the user. The NLG module produces two outputs based on the same text: one version contains basic prosodic annotations (phrase boundaries, accentuation marks) and will be further processed by the speech synthesis module, the other version consists of the same text annotated with html tags for web-based interaction, in case the user wants to explore further related information. We will briefly describe and more extensively demonstrate how we model the different aspects of a user's context (in order to annotate NLG input) and how these data influence the (output of the) NLG process. The underlying idea is that context sensitivity is a prerequisite for NLG technology to evolve with current technological developments and that a user's context encompasses more than discourse history.

2 NLG strategy in COMRIS: focus on context sensitivity

Multi-dimensional context Context is a very complex and multifaceted phenomenon which has been studied in logic for several years [3]. There is a growing awareness and interest to study it also from two other perspectives: engineering and natural language processing [4]. Within the field of NLP, mostly the linguistic (or discourse) context has been studied both for language understanding and generation. We are striving at a more global account of context, specifically for the purpose of generating natural language. In earlier experiments [5] we let NLG output vary according to the minute-by-minute evolving interests of the user. COMRIS adds another dimension: the physical context.

Thus, at least three dimensions of a user's context should influence the output of an NLG component: (a) linguistic context (the discourse model), (b) extra-linguistic context (the physical context) and (c) the user's profile. Consider the following examples (*italics* indicate context-sensitive expressions of the corresponding type):

(a) John Lewis will give a presentation on Robotics. *He* will *also* chair a panel on *the same topic.*
(b) Please note that you have to give a presentation on Monday, June 14th at 2 o'clock in Room B. Please note that you have to give a presentation *within 10 minutes* in Room B *at the other side of the building.*

[1] http://arti.vub.ac.be/~comris

(c) Isabel Baud, *who is also interested in natural language generation and wearable devices*, is currently in the same room as you. She has an interesting demo at the booth of the SIRCOM project.

We developed a simple representation formalism (see Table 1) for these three types of context information in order to annotate the propositional content provided as input to the NLG process. The text generator produces variable output according to the values of these contextual parameters.

Generation phases Ideally, context sensitivity will be taken care of in all three phases of the generation process. In COMRIS content determination and sentence planning take place outside the proper NLG component. Still, each of the three phases somehow contributes to context sensitivity.

The content of a generated utterance is mostly determined by the activities of the personal representative agents (PRA), who are exploring the virtual space in search of interesting information and encounters related to the user's interest which they have to pursue (e.g., robotics, template-based NLG). Their interactions are mainly driven by user profile information. Hence, the objects that are part of the propositional content are automatically choosen w.r.t. the hearer's interest. The outcome of PRAs activities consists of input to the NLG component (e.g., data about a panel discussion on The Future of Agents). Parts of these input structures might not be realised by the NLG component, if that suits the hearer's context better (e.g., include or omit information about the affiliation of a mentioned person might (not) be relevant for the hearer).

Sentence planning is controlled by the personal assistant agent (PA) who is on the edge of the virtual and the physical world, representing the user as a person (not one of her specific interests). Indeed, PRAs have to compete for the attention of the user with their coded message contents. The PA rules this competition, taking into account the physical context of the user but mainly by evaluating relevance, competence and performance measures (e.g., when a message, estimated very relevant by the PRA receives negative feedback from the user, the corresponding PRA's performance will decrease, thus lowering its chances in future competition for attention). Further sentence planning is carried out in the NLG component: the presence or absence of particular items in the input structure determines which rules to follow and hence which templates will be realised.

Finally, most of the context sensitive adaptations take place at the level of surface realisation through referring expressions (linguistic context), relative time and spatial expressions (physical context) and insertion of additional information (related to the user's known interests). These surface annotations are guided by the contextual annotations, contributed by the different components along the generation process (PRA: profile values; PA: physical context; NLG-discourse module: linguistic context).

Variable output based on templates We have various reasons for using templates rather than deep generation as basic technique. The expected output is canonical, i.e., it is determined by the limited set of protocols (scenes) PRAs can enter. Moreover, efficiency is a major concern, since real-time processing is required. A third argument is that modular sets of templates allow us to adapt our NLG component easily to eventual extensions of the application (new scenes). A well known limitation of templates, nl. the difficulty in reusing templates across domains and even applications holds also for our case, especially at the macro-level of sentence patterns. However, the relative ease with which templates and inputstructures can be created compensates for this limitation and we managed to reuse at least partly, some specific structures, e.g., for manipulating lists, or rendering dates, from an earlier application.

Our NLG component is developed using the TG/2 tool[2], which allows for template-based NLG as well as deep generation [6]. We currently use templates only, but as we extend our system to more scenes, we are looking for generalities that are worth implementing small subgrammars, similar to the functions in the syntactic templates of [7]. COMRIS text templates are encoded

[2] TG/2 is used in the COMRIS project under a license agreement between VUB and DFKI for the purpose of scientific research.

into TG/2 production rules in a special purpose language (TGL). A first strategy for obtainin variable output according to contextual parameters is to formulate conditions on the presence o particular context values. For instance: if the profile value denotes high interest of the hearer in th topic (of a talk to be announced), then include it in the output message. Table 1 gives an overviev of this input-output context sensitive variability for all the sub-mentioned context parameters.

Table 1. Overview of context annotation values in COMRIS

Linguistic context	**lcv (a,**	**b)**
Concept/ Instance (mentioned to user:..)	NUMeric focus on **Concept** **1**: occurred 2/more messages ago **3**: occurred 1 message ago **5**: occurred in previous message	NUMeric focus on **Instance** **1**: occurred 2/more messages ago **3**: occurred 1 message ago **5**: occurred in previous message
Extra-Linguistic context:	**d_elcv (+/- n)**	**l_elcv ("string")**
(user is in ..) time/ space	(date) 0: today (time) +1: in one hour *(time) +.10: in 10 minutes*	(location) close-by, at the other side of the building, *on the same floor, ...*
social implicature: (user is..)	**s_elcv (n)** **1..2**: standing, wandering around **3**: moving towards a goal **4**: attending an event **5**: talking to someone	
Profile Value: user's interest	**t_pv (n)**	
..in a topic:	**1..2**: .. you might be interested in **3**: .. you are interested in **5**: .. your favourite topic	
	p_pv (pnumval n	**{pqval "string"})**
..in a person:	**1..2**: .. you might want to meet **3**: .. you wanted to meet **5**: .. you absolutely wanted to meet	for dicsussion, for introduction, for lobbying, for socializing, as an expert,...

This strategy requires explicitly acquired and encoded information about the context of th hearer. However, in our experiments it became clear quite soon that the acquisition and management of such data might become a bottleneck, both from the conceptual and efficiency point o view: the NLG component has to rely on other COMRIS components to provide these data. Eacl interaction among components requires an exchange of agent messages via the COMRIS infrastructure. By putting extra burden on each of the components involved in the NLG process fron content determination (by PRA) to linguistic realisation, we might, in the worst case, endange the timely delivery of a message to the hearer.

Therefore, we are looking for alternatives using global and numeric context information, as i is handled by the COMRIS agents. For instance, we might rate the social implicature of a use at every moment, indicating how much she is involved in a social interaction (wandering around listening to a talk, talking to someone) and rate alternatives within the rule-set accordingly When the user is talking to someone, only extremely short (and important) messages should b generated, (e.g., 'Look for Isabel Baud', as an alternative for example c). Through a mechanisn of parametrization, the TG/2 tool allows to influence the order in which rules are considered, e.g. according to how much social implicature they tolerate. When the user is wandering around, th situation is less resource-bound and more elaborate messages are welcome. The social implicatur rating is a (extra-linguistic) context parameter that applies globally to the situation of the use and not to one particular aspect of the contents of the message (as with (extra)linguistic context values or profile values). Comparable to Reiter's constraints [8], we are investigating how to us such global context values which are readily available from the other COMRIS components (PA

& PRA). Being the software part of the wearable device, the PA has direct access to information about the user's environment (e.g., location near beacons, proximity to other (parrot-wearing) users), from which social implicature can be derived.

3 Discussion

In this paper we propose an NLG strategy that takes into account multiple dimensions of the user's context. The input to the NLG process is annotated with both local and global information about the context of the user. We are interested in a hybrid approach: combining local context values (e.g., indicating how much a user is interested in a particular topic of the domain, or whether a specific object has been mentioned recently to the user) with global context parameters (esp. how much the user is available for new information). We experiment with different mechanisms for producing variable output within a template based approach. Our parameters for context sensitivity are application independent, but for every application domain it has to be made explicit how they influence the output text. The COMRIS environment is an interesting test-bed for these ideas on context-sensitive NLG.

Acknowledgments We are grateful to Stephan Busemann for his advice on the use of TG/2. This research is funded by the EC as part of the COMRIS project (LTR 25500), within the Intelligent Information Interfaces[3] (I3) programme.

References

1. Not, E., Petrelli, D., Stock, O., Strapparava, C. and Zancanaro, M.: Person-oriented guided visits in a physical museum. In: Proceedings of the Fourth International Conference on Hypermedia and Interactivity in Museums (ICHIM97), Paris, (1997)
2. Van de Velde, W., Geldof, S., Schrooten, R. Competition for attention. In: Singh, M.P., Rao, A.S., and Wooldridge, M.J., (eds.) Proceedings of ATAL: 4th International Workshop on Agent Theories, Architectures and Languages, LNAI Vol. 1365. Springer-Verlag, Heidelberg (1998) 282-296.
3. McCarthey, J.: Notes on Formalizing Context. In Proceedings of 13th IJCAI conf. (1993) 555-560
4. Brézillon, P., Cavalcanti, M.: Modeling and using context. The Knowledge Engineering Review, 1997 12(4) 185-194
5. Geldof, S., Van de Velde, W.: An architecture for template based (hyper)text generation. In: Höppner, W. (ed) Proceedings of the 6th European Workshop on Natural Language Generation. Gerhard-Mercator-Universität Duisburg, Germany. (1997) 28-37
6. Busemann, S.: Best-First Surface Realization. In: Scott, D. (ed) Proceedings of the 8th Intl. workshop on Natural Language Generation, Herstmonceux Castle, University of Brigthon, UK, (1996) 101-110
7. van Deemter, K., Krahmer, E., Theune M.: Plan-based vsr. Template-based NLG: a false Opposition? In: Busemann,S. and Becker, T. (eds) Proceedings of the KI workshop on NLG: May I Speak Freely? Bonn, 1999 (this volume)
8. Reiter, E.: Shallow vs. Deep Techniques for Handling Linguistic Constraints and Optimisations. In: Busemann,S. and Becker, T. (eds) Proceedings of the KI workshop on NLG: May I Speak Freely? Bonn, 1999 (this volume)

[3] http://www.i3net.org/

Content planning and generation in continuous-speech spoken dialog systems*

Amanda J. Stent

Department of Computer Science
University of Rochester
Rochester, NY 14627
stent@cs.rochester.edu

August 12, 1999

1 Introduction

Researchers interested in constructing conversational agents that can interact naturally in relatively complex domains face a unique set of constraints. Generation must take place in real, or near-real, time. The language coverage must be extensive, and language use must be varied. A grammar-based approach can be both slow and awkward. On the other hand, it is difficult to provide the required language coverage using templates. In this paper we propose an architecture for generation that combines these two approaches, capitalizing on their strengths and minimizing their weaknesses. In the process, we attempt to answer the question, "How far can templates take us?"

2 The Situation

The TRIPS system is a multi-modal dialog system at the University of Rochester that provides a platform for research in different aspects of dialog and planning ([4]). Currently, generation in this system is done by the dialog manager and the generator. The dialog manager interprets user input and selects content for output. It passes this content to the generator as a set of role-based logical forms with associated speech acts. The generator decides which ones to produce and how to order them, and then passes them on to modality-specific generators.

In an "idealized" generation system, there would be separate components for planning intentions, semantic content, and form ([1]. Our language generator combines these last two components, and in some cases does the work of all three. It finds a rule or rules that match the speech act and content specification, and then selects from the set of meaning-equivalent strings associated with each rule. There is also a set of noun phrase rules for producing noun phrases to insert into utterances. For instance, if the logical form is for an acknowledgment, the generator may select from utterances such as "OK" and "Fine". For location question answers, it may have only one utterance, "There are NP" in which "NP" can be replaced by e.g. "five people at Calypso". For most simple utterances (acknowledgments, indications of lack of understanding or reference failures) and a limited set of more complex ones, this is sufficient. As the system begins

*This work was supported by ONR research grant N00014-95-1-1088, U.S. Air Force/Rome Labs research contract no. F30602-95-1-0025, NSF research grant no. IRI-9623665 and Columbia University/NSF research grant no. OPG: 1307

A	41	And so [breath] I think we need to send a we obviously need to send an ambulance to Marketplace.
	42	We should then send that ambulance to Highland.
B	43	Mm-hm.
A	44	Um so the problem is th- the six people at the airport.
	45	Um we can do the helicopter from the airport to Strong.
	46	Those a- are in fact the only two places that you can do that.
B	47	Right.
A	48	Um [breath] so here's the thing.
	49	We can we we can either uh
	50	I guess we have to decide how to break up this.
	51	We can make three trips with a helicopter a-
B	52	So I guess we should send one ambulance straight off to Marketplace right now right?

Figure 1: Dialog extract

to take more initiative and the domain becomes more complex (answering wh-questions, making statements, asking questions), this approach becomes more difficult to maintain.

The output of generation is a string or set of strings that are sent to the Truetalk speech synthesizer and displayed on the screen, and displays including maps, drawings on maps and charts. Currently, push-to-talk regulates the turn-taking, but we hope soon to move to continuous speech.

Our goal is to develop a conversational agent capable of interacting naturally in task-oriented multi-modal dialog situations. To give us an idea of the kinds of interaction required, we have collected a corpus of twenty mixed-initiative, task-oriented human-human dialogs in a complex domain. Our study of these dialogs informs our hypotheses about how to produce natural interactions.

3 Dialog

In the following discussion we will use the extract in figure 1, which comes from a dialog in our corpus, as a reference point.

It is important to distinguish between the plan for the dialog as a whole, and the plan for the current utterance or turn. The dialog plan is constructed as a byproduct of the agents' collaboration. It is impossible to construct the whole dialog plan at the beginning; planning must be incremental and take place in real-time. Re-planning may have to occur at any point, to deal with interruptions or new information from the world. By contrast, the plan for the current utterance can usually be specified in full (although even here some researchers prefer to interleave planning and execution [7]). Because the dialog plan is incomplete, however, the plan for any individual utterance will necessarily be made on the basis of incomplete information.

Because dialog is collaborative behavior, there are two kinds of intentions behind the production of individual utterances: task/domain-related intentions that contribute to the overall dialog plan; and intentions related to maintaining the collaboration (see figure 2).

Some utterances are related to the topic of the dialog, for instance contributing to the solution of a task. The intentions behind these may come from a task or domain model ([2, 5]), or from a model of rhetorical structure ([6]). The intentions behind utterance 45 probably come from the domain model, while utterance 44 provides **motivation** for utterance 45. These utterances also have semantic content in addition to the speech act, such as references to specific objects. Finally, the form of these utterances matter. For instance, the "then" in utterance 42 is crucial to identification of the **sequence** relation that holds between utterances 41 and 42.

Other utterances (e.g. turn-taking and grounding utterances) maintain the collaboration.

Type of dialog act	Source of intentions	Examples
turn-taking	maintains collaboration	"Um", "Wait a minute"
grounding	maintains collaboration	"Okay", "Well...."
primary acts	furthers task	"Send the A train."
secondary acts	support, coherence	"because it's faster."

Figure 2: The different types of dialog acts

These communicative actions are intentional, but are not part of the overall dialog plan. Often, to produce them one can simply select one of several conventional utterances that satisfy the intention . There is no need to plan semantic content or form for these utterances. For example, utterance 43 is an utterance that performs grounding only.

Looking at the example dialog, we can see that planning utterances does not consist simply of choosing individual dialog acts and then realizing them; multiple dialog acts can be performed by one utterance or a single dialog act may be realized over several utterances ([8]). For example, a *release-turn* act can be performed by performing an *info-request*. We hope that as we annotate our dialog corpus, we will gain insights into the complex interactions between different aspects of content planning and generation. For instance, how frequently do agents reuse surface forms? How do agents decide when they need not plan a grounding act? How do agents combine different dialog acts, and what surface forms do they use to signal these combinations? In what circumstances will agents generate utterances that contribute to rhetorical structure, and when will they limit how much they say?

4 Proposed Architecture

We propose to think of the three stages of generation (planning intentions, planning content and planning form) as three different dimensions along which planning[1] can occur, possibly simultaneously ([7, 3]). The planning of intentions generally consists of selecting intentions from different sources such as interpretation and the agent's internal agent model, and ensuring that none of the selected intentions conflict or are redundant. It usually takes place as part of dialog management. The planning of content is what is more usually referred to as strategic generation, and the planning of form is tactical generation.

If planning need occur along only one dimension, then templates can be effectively used. Grounding and turn-taking acts involve the planning of intentions only. The form can be selected from a set of conventional forms; to try to construct content and generate these surface forms using a grammar adds nothing to the result, and may limit the variability of language use. It can sometimes involve completely unnecessary processing (think of generating the surface form "I heard you" from the discourse act *acknowledge*). Also, turn-taking and grounding acts often begin a turn, so generating these acts quickly can give a conversational agent time to produce other acts that may involve more processing. Therefore, we can use templates to generate these acts.

If planning needs to occur along more than one dimension, it may be better to use a grammar. Otherwise, a template will probably be needed for each combination of, say, intention and content or content and form (see figure 3).

Those utterances that speakers produce to fulfill intentions arising directly from the domain or the task being solved (*primary intentions*) often have content that must be expressed. The form

[1]In this paper, *planning* is any kind of non-trivial processing.

Dimensions that involve planning	Type of dialog act	Grammar/ templates
I	turn-taking, grounding	templates
C		
F		
I, C	some primary intentions	grammar
I, F		
C, F		
I, C, F	some primary intentions, secondary intentions	grammar

Figure 3: Generating different types of dialog acts: dimensions along which planning may occur

may or may not be important. These utterances should be generated using a grammar, unless there is a very limited set of kinds of utterances that can be produced. There is nothing to gain from using templates because there is no way to "skip" stages of processing, and with a grammar greater language coverage can be obtained.

Other utterances are produced primarily to complete an argumentation act, for instance to provide justification for something (we will call these *secondary intentions*). Their production involves the planning of intentions, semantic content and surface form. In particular, the form may determine whether these acts are seen as coherent in the dialog. These utterances should also be generated using a grammar.

These four kinds of dialog acts account for only three of the seven possible combinations of dimensions along which planning may occur (see figure 3). We are unable to think of examples of utterances for the other four possibilities[2], so we have left them blank, but we believe the same reasoning could be used in these cases.

To summarize, we believe that templates are best used when it is possible to eliminate stages of processing (e.g. to go directly from intentions to form), and when speed is necessary. Otherwise, we think a grammar should be used for tactical generation, especially where broad language coverage is needed.

At this point, we may conclude that we have obtained a good architecture for generation for dialog. Intentions, represented as dialog acts ([8]), and associated content come to the generator from the agent's internal agent model (which can reason about the task, the domain, and rhetorical structure), or from the process of interpretation. Each kind of dialog act proceeds through a different path in the architecture. The generation of turn-taking and grounding acts happens quickly, via templates. There is the coverage of a grammar for producing the more "multi-dimensional" acts.

If a grammar that provides incremental output is used, the behavior of the agent will change. One might expect to see fewer utterances that perform only grounding at the start of turns. There will also probably be repairs; the modules producing incremental output would provide the repairs but, if there are pauses, turn-keeping utterances could be interleaved with the incremental output from the other modules.

Unfortunately, this architecture is a little too simple. As we said earlier, many utterances realize multiple dialog acts. For instance, questions can be both *info-request* and *release-turn* acts. We cannot just send the dialog acts through their respective paths without risking over-generation. We have a very preliminary solution for this. The generator maintains a set of sets of intention by content pairs, prioritized mostly by recency (we'll call this the *intention-set*). Each set is sent to all the generation modules. The output from each module is a a surface form and a set of intentions fulfilled by that form. A gate-keeper at the end removes intentions from the intention-set as they are fulfilled. It can also add sets of intentions to the memory, for instance to keep the turn or

[2]If the domain is simple enough that everything to be generated is an *inform*, that might be a case where only content, or maybe only content and form, need be planned. However, few domains are that simple.

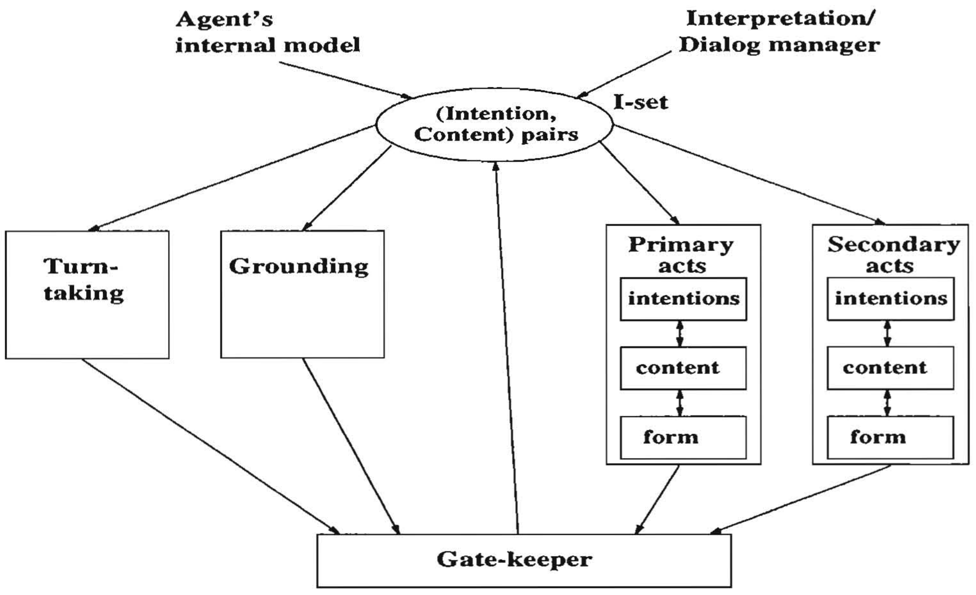


Figure 4: Proposed generator architecture

if the agent is interrupted. Finally, it can minimize over-generation by selecting which results to produce, if it gets simultaneous results that satisfy the same intentions.

For example, imagine a user has just made a statement to the agent. The agent wants to acknowledge part of the statement (grounding) and ask a question about another part. So the memory looks like:
{{*take-turn, acknowledge(Utt1), info-request(Content)*}}
(items with initial capital letters are variables).
This set gets passed to all modules. The turn-taking module returns "Uh" for *take-turn* and the grounding module returns "Okay" for *take-turn* and *acknowledge(Utt1)*. The gate-keeper therefore removes *take-turn* and *acknowledge(Utt1)* and produces "Okay". If a pause of more than, say, half-a-second ensues, the gate-keeper might add the set {*keep-turn*} to the memory which will feed it to the various modules. However, happily the gate-keeper quickly receives a result for *info-request(Content)* which it produces, removing that intention (and therefore the whole set) from the memory.

This architecture is given in figure 4. We have not yet implemented it, but we believe it may be possible to combine the turn-taking and grounding modules into one, and the primary and secondary act modules into one. This would especially help with reasoning about argumentation acts.

Real-time generation is very important in the context of dialog. We have observed that if users have to wait for a response, they may begin to hyper-articulate, resort to saying only one word per utterance, or otherwise begin to use unnatural interactions. In a continuous-speech system in particular, this architecture would be most effective if the interpretation component could produce incremental output, thus allowing the system to, for instance, provide appropriate and timely back-channels.

Of course, agents have many intentions when interacting, among them social intentions such as politeness, and other "global" intentions such as efficiency. We have not discussed how these types of intentions could be used in this architecture. We believe that they could simply be included as constraints on the generation process, as they are in some text-based generators.

We should point out that there is nothing in the architecture that requires that templates or a grammar need be used in any module; either of these, or other forms of generation (finite state models, statistical generators) can be used.

This architecture allows for different levels of processing, incremental generation, and fast generation in some cases. It also allows us to combine different types of generators into one component. We believe it, or something like it, will permit natural interaction in the context of conversational agents.

5 Conclusion

We have highlighted the unique difficulties of performing generation for free-flowing task-oriented dialog and the possibilities inherent in using an approach to generation that combines the use of templates with the use of a grammar and planning. We have also classified some of the ways templates can be used in strategic and tactical generation for dialog.

These are our initial hypotheses based on a preliminary examination of our data. We hope to soon be able to confirm or deny them, and point out any complicating factors of which we become aware during further data analysis and preliminary system development.

References

[1] M. Bordegoni, G. Faconti, S. Feiner, M. Maybury, T Rist, S. Ruggieri, P. Trahanias, and M. Wilson. A standard reference model for intelligent multimedia presentation systems. *Computer Standards and Interfaces*, 18(6, 7):477–496, December 1997.

[2] J. Chu-Carroll and S. Carberry. Collaborative response generation in planning dialogues. *Computational Linguistics*, 24(3):355–400, 1998.

[3] K. De Smedt, H. Horacek, and M. Zock. Architectures for natural language generation: Problems and perspectives. In *Trends in Natural Language Generation: An Artificial Intelligence Perspective*, pages 17–46. Springer-Verlag, Berlin, Germany, 1996.

[4] G. Ferguson and J. Allen. TRIPS: an intelligent integrated problem-solving assistant. In *Proceedings of the fifteenth national conference on artificial intelligence (AAAI-98)*, Madison, WI, 1998.

[5] K. Lochbaum. A collaborative planning model of intentional structure. *Computational Linguistics*, 24(4):525–572, December 1998.

[6] W. Mann and S. Thompson. Rhetorical structure theory: a theory of text organisation. In L. Polanyi, editor, *The structure of discourse.* Ablex, Norwood, NJ, 1987.

[7] N. Reithinger. POPEL – a parallel and incremental natural language generation system. In C. L. Paris, W. R. Swartout, and W. C. Mann, editors, *Natural Language Generation in Artificial Intelligence and Computational Linguistics*, pages 179–199. Kluwer Academic Publishers, Boston, MA, 1991.

[8] D. Traum and E. Hinkelman. Conversation acts in task-oriented spoken dialogue. *Computational Linguistics*, 18(3):575–599, 1992.

Output Generation in a Spoken Dialogue System

Lena Santamarta
Dept. of Computer and Information Science
Linköping University, S-581 83 Linköping, Sweden
lensa@ida.liu.se

June 29, 1999

1 Introduction

Spoken dialogue systems are finding their way to the public at large, e.g. as simple service systems, and have then to handle a range of users that differ in many ways: in age, educational status, computer experience, and others. A public dialogue system should work for all users. The users may use such a system sporadically. This requires that the system is robust in all its parts, as a failure in the first interaction could discourage the user from using the system again. Public systems do not afford being dependent on the users' learning. We believe that some of the factors that influence the acceptability of a system are the flexibility and the quality of the output.

This paper presents some basic assumptions of work in progress which aims to develop the output capacities of the dialogue system built at Linköping University, the LinLin system [6]. The LinLin system was first developed to cope with written NL interaction and is now being developed towards a speech-in-speech-out multimodal system. However, the work presented in this paper deals with speech-only interaction. The goal is to achieve a flexible generator that allows the interaction to take place on the user's terms.

The LinLin system is a general dialogue framework for simple service systems that relies on the concept of sub-languages to cope with human-machine-interaction in natural language. For managing an information-seeking dialogue it is enough to model the sub-language of human-computer-interaction and the sub-language of the domain the application deals with. The dialogue is handled by a dialogue grammar that models it in initiative-response units. Empirical studies show that the coherence relation between turns in a human-computer-dialogue is rather simple. Users talk about the same object or the same properties [6]. Currently we are working on a public transport information domain.

Spoken dialogue systems have special requirements on generation; the generated answer has to be short, only the most suitable information can be presented, and it has to be presented in such a way that the relevance and importance of each piece of information are clearly mirrored in the output.

A dialogue system consists of several modules, here we will focus on the generation module and the dialogue manager that controls the whole system and the interaction with the user. The generator is thus a slave of the dialogue manager.

2 The Input and Output

The dialogue manager provides the generation module with the content of the next utterance to be generated and the communicative goal this utterance has to fulfil. Beside this, it is necessary that the generator has access to information about the words chosen by the user and the focus of the user's turn. The dialogue tree built by the dialogue manager is also available for the generator. The classical generation task of deciding "what to say" is consequently handled by the dialogue manager.

The generator outputs a tagged text adapted for a speech synthesis system. In the last years there have been initiatives to specify and standardise such a mark-up language [10]. The main idea when generating marked text is to transfer the information about syntax, semantics, and topic structure that the generator has to the speech synthesis [7].

The output text is tagged with pauses and boundary types indicating the prosodic units of the utterances. Words are tagged for prominence level depending on their new/given status, part-of-speech and syntactical function, as well as if they are part of a contrastive or emphatic construction. (For a description of the phonetic realisation of these features see e.g. [3], [4], [5]).

3 Generation in a Spoken Dialogue System

In a dialogue system the length of the generated text is rather short, namely a turn. A turn can consist of one word up to a couple of sentences. However, the versatility of the channel accentuates the need of proper structuring the information, syntactically and prosodically. The modelling of contrast and emphasis, of the topical structure and the new-given relationship is very important for the intelligibility of the synthesised speech.

Generation in a dialogue system consist of two different kinds of processes, there is a context dependent part that varies due to the state of the dialog and the user's turn and there is a linguistic part that do not depend on the turn or even the domain. The first part includes the steps of ordering the pieces of information to be presented, taking care of lexicalization, and marking the context dependent information status of constituents. A concept's information status affects its prominence level and depends on whether the concept is new or given in the discourse and on its relevance in the fulfilment of the system's next dialogue move. In the second part the linguistic surface form has to be created with the corresponding default prosodic features. Obviously, these correspond to the "sentence planner" and the "linguistic realiser" of NLG architecture [9]. In a spoken interaction environment, however, "utterance planner" should be a better name as there is not always complete sentences that are generated.

3.1 The Utterance Planner

To make the system answer co-operative, the utterance planner's main concern is to tailor it against the user's information needs and linguistic preferences. The user's information needs are assumed to be new information and relevant information. What is new information can be inferred from the user's question and the dialogue history. What is relevant information in a domain is defined by domain heuristics developed from the initial empirical studies. While the derivation of what is new information is a procedure that can be reused from one application to another, the relevance notion is strictly application dependent. To tailor the answer to the linguistic preferences of the user the system should adapt to the lexical choices of the user.

Some of the problems the utterance planner has to deal with are how to order the presentation of information and how to mark for prominence. In initial empirical studies we have seen that new and relevant information comes early in the answer and is prosodically prominent. To do this the system must be able to present the same response from the background system, i.e. the same content, in different forms depending on what was the focus of the user's question.

Consider for example the two interactions below [1]:

EXAMPLE 1:
U: *How do I get from the railway station to IKEA tomorrow round ten?*
S: *Take bus* **210** *from the railway station at* **ten o'clock**. *It will be at IKEA at* **twenty past ten.**

EXAMPLE 2:
U: *When is the next bus going from the railway station to IKEA?*
S: *at* **ten o'clock,** *the next* **210** *leaves at ten o'clock from the railway station. It will be at IKEA at* **twenty past ten.**

Both questions have the same response from the background system, namely:

Bus	Resecentrum	IKEA
210	10:00	10:20

In both cases the dialogue manager wants the generator to inform the user about this data that fitted as a response to the questions. However, in the first case the system recognises the turn as a request for a "route", i.e. a description of how to use the buses to go from A to B. And in the second case the system recognises the request as being for a "trip", i.e. a request for information about a special bus and departure time. The raw information content of the two answers is the same, but the surface form is different because the users requested different information. In both cases the arrival time comes last, this is not due to the new/given status of it but to the domain heuristics that states that arrival time is only relevant information when it is explicitly asked for. The phrase "ten o'clock" will be prominent because it is relevant information in the answer, though not altogether new information. This also depends on the domain heuristics. Departure time is relevant and important information, otherwise the user may miss the bus and the service be useless.

The lexicalization step should follow an adaptation strategy. Many studies have shown that users usually adapt to the vocabulary of the computer, changing their lexical choice for the one of the system, specially in respect to content word as verbs and nouns (see e.g. [2]). However, in some domain the users usually do not use the "official" names of the entities but other types of referents, this is true about the bus traffic domain [1]. In the examples above the user refers to "the railway station" while the bus stop is really called "Resecentrum". Suppose the user never heard of "Resecentrum", an answer containing that word instead of the one chosen by the user would make no sense for the user. To start the answer telling the user what the official name of the bus stop is goes against the principles for ordering stated

[1]**Bold** indicates that the word or phrase is prominent.

before. So we choose to let the system adapt to the user's lexical choice. This strategy is also used to choose between linguistic synonyms when possible. This yields a co-operative dialogue system and eases the process of lexical choice.

The utterance planner has to mark the output for the context dependent information status. Concepts are marked as new information, given information or the intermediate given-by-semantic-relation information, because in speech new concepts are prominent while given concepts are not, and items that are new to the discourse may be de-accentuated or have a special realisation if they are given by the context or through a hyponym relationship with an already mentioned concept. Concepts are also marked if they are part of an emphatic or contrastive construction.

In order to produce a naturally sounding utterance the utterance planner, as sentence planner for written language, takes care of aggregation and pronominalization as well as the phases described above. However, those process should not be different in a spoken language context.

3.2 The Linguistic Realiser

The linguistic realiser takes the lexicalized utterance plan and has to output the surface form. The surface realisation includes not only making the correct morphological and syntactical forms but dealing with the information status markers. The realiser has to accomplish the prosodic marking by applying surface form dependent prosodic rules.

The syntactic structure of the utterance is used to define phrase and sentence boundaries that are marked in speech by pauses and intonation contours. Within a phrase, pauses are placed between prominent words. In sentences with all new information the constituents have different level of prominence depending on their syntactical function, non-prominent content words sometimes have to be marked with weak intonation patterns depending on the distance to the prominent word [3]. Contrast and emphasis are signalled in speech both syntactically and prosodically. The linguistic realiser has to be able to handle the markers put there by the utterance planner and elaborate them to create a full prosodic description of the utterances.

4 Templates vs. Free Choice

To discuss re-use of generation resources when it comes to the Swedish language is pointless, there are no such available. So, the key question is whether to put a big effort in developing templates that are flexible (or many) enough to accomplish the communication strategy presented above or to build a new system for the realisation of Swedish, alternatively to build a Swedish variant of resources available for other languages.

Besides the well-known pros and cons of templates and canned text describe by many authors (see e.g. [8]) in a dialogue system environment there are extra cons that are worth to point out.

- As pointed out above the surface form of the answer depends on the question in many different ways and the same information from the background system can be presented in different ways depending on the user's question and the dialogue history. This would demand a huge amount of templates to be built.

- In a speech-only environment, due to the restriction of the channel, only the requested information should be presented. Using templates we would need different templates for the different amount of information to be presented.
- In order to adapt to the user's vocabulary, the templates need to have variables in parts where they usually do not, thus increasing the choice points.
- Templates and canned texts may lead to a mismatch between system's language understanding and production capacities.

The last point is not a consequence of using templates and canned texts, but a serious risk as dialogue systems usually are developed by several different researches working on different parts of the system. Dialogue systems are often developed to "understand" the many different ways of expression that the users may choose while interacting. Those alternatives are then mapped into an internal representation that triggers a query to the background system and an answer to the user. Thus, there is a many-to-one mapping that could lead to a system that is able to understand expressions it does not use. The system's utterances are then monotonous and the user may feel that the system do not accept his way of expression.

Canned text and the predefined parts of templates may consist of words and syntactical structures the system does not have in the understanding module. Specially in larger text for help, or for presentations and descriptions of objects. Then the system could use words and structures that it cannot understand itself. This is indeed more "dangerous" than the previous because it can lead to utterances as *I'm sorry. I don't understand the word "sorry".* To avoid this problem the lexicon of the generator has to be shared with the rest of the system. In any case, a mismatch between the production and the perception of the system may confuse the user. We do commonly assume that our interlocutors understand the words and structures they use, and that they can say everything they understand.

In order to avoid this kind of communicative mismatches, the developers have to put much attention when building and enhancing the system, specially in putting all words and expressions from the produced text into the interpretation module. This special attention have to be added to the costs of developing template based dialogue systems.

The answer to the key question posed above seems to be not to use templates, but is it then to use a general free choice system? As pointed above there is no such system for Swedish that we may re-use, so the system has to be built. And as this work deals with spoken interaction the system will be tailored to that kind of language and will not be re-usable for the generation of larger written text. This means that the amount of superfluous grammatical coverage will be limited. As described above, the utterance planner is more domain and application dependent than the linguistic realiser, the solution seems to be to build a general linguistic realiser for spoken human-computer-interaction Swedish and an easily customisable framework for the utterance planning component.

As no spoken dialogue system for Swedish use the kind of architecture proposed in this paper for generation, we can not say with confidence what kind of problems the Swedish language or the grammar of spoken language may cause. What we can say is that the templates based systems used in the systems that are available (public or at research labs) do not allow the kind of communication strategy we think is necessary for public systems.

5 Discussion

We have presented the requirements spoken dialogue systems have on generation in order to produce an efficient and co-operative interaction with the user. Co-operative answer that are tailored towards the user's information needs and linguistic preferences are important in making public dialogue systems that do not require learning or computer experience to be useful and efficient.

An architecture as proposed above, a customisable domain dependent planner and a general realiser for the human-computer-interaction sub-language, will be suitable for spoken interaction in the kinds of applications the LinLin system works with, i.e. information-seeking dialogues for simple services. The basic ideas could also be implemented (and the utterance planner enhanced) for more sophisticated dialogue system architectures.

References

[1] Flycht-Eriksson, A., Jönsson, A.: "A spoken dialogue system utilizing spatial information". Proceedings of ICSLP'98, Sydney, Australia, 1998.

[2] Gustafson, J., Larsson, A., Carlson, R., Hellman, K.: "How do System Questions Influence Lexical Choices in User Answers?". In: Proceedings of EUROSPEECH'97, Rhodes, Greece, 1997

[3] Horne, M.: Towards a quantified, focus-based model for synthesizing English sentence intonation. Lingua 75, Lund University, 1988

[4] Horne, M., Filipsson, M.: "Computational modelling and generation of prosodic structure in Swedish". In: Proceedings of ICPhS'95, Stockholm, Sweden, 1995.

[5] Horne, M., Hansson, P., Bruce, G., Frid, J.: "Prosodic Correlates of Information Structure in Swedish Human-Human Dialogues". Forthcoming in: Proceedings of EUROSPEECH'99, Budapest, Hungary, 1999.

[6] Jönsson, A.: Dialogue Management for Natural Language Interfaces, An Empirical Approach. Linköping Studies in Science and Technology, Dissertations, No. 312, 1993

[7] Pan, S., McKeown, K.R.: "Integrating Language Generation with Speech Synthesis in a Concept to Speech System". In: Alter, K., Pirker, H., Finkler, W. (eds): Concept to Speech Generation Systems, ACL, Madrid, Spain, 1997

[8] Reiter, E.: "NLG vs. Templates". In Proceedings of the Fifth European Workshop on Natural-Language Generation, Leiden, The Netherlands, 1995.

[9] Reiter, E., Dale, R.: "Building Applied Natural-Language Generation Systems". Journal of Natural-Language Engineering, No 3, 1997.

[10] Sproat, R., Taylor, P., Tanenblatt, M., Isard, A.: "A markup language for text-to-speech synthesis". Proceedings of EUROSPEECH'97, Rhodes, Greece, 1997

Veröffentlichungen des DFKI

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse oder (so sie als per ftp erhaeltlich angemerkt sind) per anonymous ftp von ftp.dfki.uni-kl.de (131.246.241.100) im Verzeichnis pub/Publications bezogen werden. Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all published papers so far are obtainable from the above address or (if they are marked as obtainable by ftp) by anonymous ftp from ftp.dfki.uni-kl.de (131.246.241.100) in the directory pub/Publications.
The reports are distributed free of charge except where otherwise noted.

DFKI Research Reports

1999

RR-99-04
Gero Vierke, Christian Ruß
The Matrix Auction: A Mechanism for the Market-Based Coordination of Enterprise Networks
11 pages

RR-99-03
Christian Gerber, Jörg Siekmann, Gero Vierke
Holonic Multi-Agent Systems
42 pages

RR-99-02
*Michael Schillo, Jürgen Lind, Petra Funk, Christian Gerber,
Christoph Jung*
SIF - The Social Interaction Framework
System Description and User's Guide to a Multi-Agent System Testbed
30 pages

RR-99-01
Jürgen Lind, Stefan Philipps
Ein System zur Definition und Ausführung von Protokollen für Multi-Agentensysteme
61 Seiten

1998

RR-98-04
Bernd Kiefer, Hans-Ulrich Krieger
A Bag of Useful Techniques for Efficient and Robust Parsing
9 pages

RR-98-03
Heiko Mantel
Developing a Matrix Characterization for $\mathcal{MELL}$
59 pages

RR-98-02
Klaus Fischer, Christian Ruß, Gero Vierke
Decision Theory and Coordination in Multiagent Systems
134 pages

RR-98-01
Christoph G. Jung, Klaus Fischer
Methodological Comparison of Agent Models
58 pages

1997

RR-97-08
Stefan Müller
Complement Extraction Lexical Rules and Argument Attraction
14 pages

RR-97-07
Stefan Müller
Yet Another Paper about Partial Verb Phrase Fronting in German
26 pages

RR-97-06
Stefan Müller
Scrambling in German – Extraction into the *Mittelfeld*
24 pages

RR-97-05
Harald Meyer auf'm Hofe
Finding Regions of Local Repair in Hierarchical Constraint Satisfaction
33 pages

RR-97-04
Serge Autexier, Dieter Hutter
Parameterized Abstractions used for Proof-Planning
13 pages

RR-97-03
Dieter Hutter
Using Rippling to Prove the Termination of Algorithms
15 pages

RR-97-02
Stephan Busemann, Thierry Declerck, Abdel Kader Diagne, Luca Dini,
Judith Klein, Sven Schmeier
Natural Language Dialogue Service for Appointment Scheduling Agents
15 pages

RR-97-01
Erica Melis, Claus Sengler
Analogy in Verification of State-Based Specifications: First Results
12 pages

1996

RR-96-06
Claus Sengler
Case Studies of Non-Freely Generated Data Types
200 pages

RR-96-05
Stephan Busemann
Best-First Surface Realization
11 pages

RR-96-04
Christoph G. Jung, Klaus Fischer, Alastair Burt
Multi-Agent Planning
Using an *Abductive*
EVENT CALCULUS
114 pages

RR-96-03
Günter Neumann
Interleaving
Natural Language Parsing and Generation
Through Uniform Processing
51 pages

RR-96-02
E.André, J. Müller , T.Rist:
PPP-Persona: Ein objektorientierter Multimedia-Präsentationsagent
14 Seiten

RR-96-01
Claus Sengler
Induction on Non-Freely Generated Data Types
188 pages

1995

RR-95-20
Hans-Ulrich Krieger
Typed Feature Structures, Definite Equivalences, Greatest Model Semantics, and Nonmonotonicity
27 pages

RR-95-19
Abdel Kader Diagne, Walter Kasper, Hans-Ulrich Krieger
Distributed Parsing With HPSG Grammar
20 pages

RR-95-18
Hans-Ulrich Krieger, Ulrich Schäfer
Efficient Parameterizable Type Expansion for Typed Feature Formalisms
19 pages

RR-95-17
Hans-Ulrich Krieger
Classification and Representation of Types in TDL
17 pages

RR-95-16
Martin Müller, Tobias Van Roy
Title not set
0 pages

Note: The author(s) were unable to deliver this document for printing before the end of the year. It will be printed next year.

RR-95-15
Joachim Niehren, Tobias Van Roy
Title not set
0 pages

Note: The author(s) were unable to deliver this document for printing before the end of the year. It will be printed next year.

RR-95-14
Joachim Niehren
Functional Computation as Concurrent Computation
50 pages

RR-95-13
Werner Stephan, Susanne Biundo
Deduction-based Refinement Planning
14 pages

RR-95-12
Walter Hower, Winfried H. Graf
Research in Constraint-Based Layout, Visualization, CAD, and Related Topics: A Bibliographical Survey
33 pages

RR-95-11
Anne Kilger, Wolgang Finkler
Incremental Generation for Real-Time Applications
47 pages

RR-95-10
Gert Smolka
The Oz Programming Model
23 pages

RR-95-09
M. Buchheit, F. M. Donini, W. Nutt, A. Schaerf
A Refined Architecture for Terminological Systems: Terminology = Schema + Views
71 pages

RR-95-08
Michael Mehl, Ralf Scheidhauer, Christian Schulte
An Abstract Machine for Oz
23 pages

RR-95-07
Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt
The Complexity of Concept Languages
57 pages

RR-95-06
Bernd Kiefer, Thomas Fettig
FEGRAMED
An interactive Graphics Editor for Feature Structures
37 pages

RR-95-05
Rolf Backofen, James Rogers, K. Vijay-Shanker
A First-Order Axiomatization of the Theory of Finite Trees
35 pages

RR-95-04
M. Buchheit, H.-J. Bürckert, B. Hollunder, A. Laux, W. Nutt,
M. Wójcik
Task Acquisition with a Description Logic Reasoner
17 pages

RR-95-03
Stephan Baumann, Michael Malburg, Hans-Guenther Hein, Rainer Hoch,
Thomas Kieninger, Norbert Kuhn
Document Analysis at DFKI
Part 2: Information Extraction
40 pages

RR-95-02
Majdi Ben Hadj Ali, Frank Fein, Frank Hoenes, Thorsten Jaeger,
Achim Weigel
Document Analysis at DFKI
Part 1: Image Analysis and Text Recognition
69 pages

RR-95-01
Klaus Fischer, Jörg P. Müller, Markus Pischel
Cooperative Transportation Scheduling
an application Domain for DAI
31 pages

1994

RR-94-39
Hans-Ulrich Krieger
Typed Feature Formalisms as a Common Basis for Linguistic Specification.
21 pages

RR-94-38
Hans Uszkoreit, Rolf Backofen, Stephan Busemann, Abdel Kader Diagne,
Elizabeth A. Hinkelman, Walter Kasper, Bernd Kiefer, Hans-Ulrich Krieger,
Klaus Netter, Günter Neumann, Stephan Oepen, Stephen P. Spackman.
DISCO–An HPSG-based NLP System and its Application for Appointment Scheduling.
13 pages

RR-94-37
Hans-Ulrich Krieger, Ulrich Schäfer
TDL - A Type Description Language for HPSG, Part 1: Overview.
54 pages

RR-94-36
Manfred Meyer
Issues in Concurrent Knowledge Engineering. Knowledge Base and Knowledge Share Evolution.
17 pages

RR-94-35
Rolf Backofen
A Complete Axiomatization of a Theory with Feature and Arity Constraints
49 pages

1995

TM-95-04
Klaus Schmid
Creative Problem Solving
and
Automated Discovery
— An Analysis of Psychological and AI Research —
152 pages

TM-95-03
Andreas Abecker, Harold Boley, Knut Hinkelmann, Holger Wache,
Franz Schmalhofer
An Environment for Exploring and Validating Declarative Knowledge
11 pages

TM-95-02
Michael Sintek
FLIP: Functional-plus-Logic Programming
on an Integrated Platform
106 pages

TM-95-01
Martin Buchheit, Rüdiger Klein, Werner Nutt
Constructive Problem Solving: A Model Construction Approach towards Configuration
34 pages

1994

TM-94-05
Klaus Fischer, Jörg P. Müller, Markus Pischel
Unifying Control in a Layered Agent Architecture
27 pages

TM-94-04
Cornelia Fischer
PAntUDE - An Anti-Unification Algorithm for Expressing Refined Generalizations
22 pages

TM-94-03
Victoria Hall
Uncertainty-Valued Horn Clauses
31 pages

TM-94-02
Rainer Bleisinger, Berthold Kröll
Representation of Non-Convex Time Intervals and Propagation of Non-Convex Relations
11 pages

TM-94-01
Rainer Bleisinger, Klaus-Peter Gores
Text Skimming as a Part in Paper Document Understanding
14 pages

DFKI Documents

1999

D-99-01
Tilman Becker, Stephan Busemann
May I Speak Freely? Between Templates and Free Choice in Natural Language Generation Workshop at *the 23rd German Annual Conference for Artificial* Intelligence (KI '99), Bonn 14.-15. September 1999
69 pages

1998

D-98-03
Stephan Busemann, Karin Harbusch, Stefan Wermter(Hrsg.)
Hybride konnektionistische, statistische und regelbasierte Ansätze zur Verarbeitung natürlicher Sprache Workshop auf der 21. Deutschen Jahrestagung für Künstliche Intelligenz, Freiburg, 9.–10. September 1997
75 Seiten

D-98-02
Andreas Abecker, Ansgar Bernardi, Knut Hinkelmann , Otto Kühn,
Michael Sintek
Techniques for Organizational Memory Information Systems
66 pages

D-98-01
Stephan Baumann, Jürgen Lichter, Michael Malburg, Heiko Maus,
Harald Meyer auf'm Hofe, Claudia Wenzel
Architektur für ein System zur Dokumentanalyse im Unternehmenskontext Integration von Datenbeständen, Aufbau- und Ablauforganisation
76 Seiten

1997

D-97-08
Christoph G. Jung, Klaus Fischer, Susanne Schacht
Distributed Cognitive Systems
Proceedings of the VKS'97 Workshop
50 pages

D-97-07
Harold Boley, Bernd Bachmann, Christian Blum, Christian Embacher,
Andreas Lorenz, Jamel Zakraoui
PIMaS:
Ein objektorientiert-regelbasiertes System zur Produkt-Prozeß-Transformation
45 Seiten

D-97-06
Tilman Becker, Stephan Busemann, Wolfgang Finkler
DFKI Workshop on Natural Language Generation
67 pages

D-97-05
Stephan Baumann, Majdi Ben Hadj Ali, Jürgen Lichter, Michael Malburg,
Harald Meyer auf'm Hofe, Claudia Wenzel
Anforderungen an ein System zur Dokumentanalyse im Unternehmenskontext
— Integration von Datenbeständen, Aufbau- und Ablauforganisation
42 Seiten

D-97-04
Claudia Wenzel, Markus Junker
Entwurf einer Patternbeschreibungssprache
für die Informationsextraktion
in der Dokumentanalyse
24 Seiten

D-97-03
Andreas Abecker, Stefan Decker, Knut Hinkelmann, Ulrich Reimer
Proceedings of the Workshop „Knowledge-Based Systems for Knowledge Management in Enterprises“ 97
167 pages

D-97-02
Tilman Becker, Hans-Ulrich Krieger
Proceedings of the Fifth Meeting on Mathematics of Language (MOL5)
168 pages

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

D-97-01
Thomas Malik
NetGLTool Benutzeranleitung
40 Seiten

1996

D-96-07
Technical Staff
DFKI Jahresbericht 1995
55 Seiten

Note: This document is no longer available in printed form.

D-96-06
Klaus Fischer (Ed.)
Working Notes of the KI'96 Workshop on Agent-Oriented Programming and Distributed Systems
63 pages

D-96-05
Martin Schaaf
Ein Framework zur Erstellung verteilter Anwendungen
94 pages

D-96-04
Franz Baader, Hans-Jürgen Bürckert, Andreas Günter, Werner Nutt (Hrsg.)
Proceedings of the Workshop on Knowledge Representation and Configuration WRKP'96
83 pages

D-96-03
Winfried Tautges
Der DESIGN-ANALYZER - Decision Support im Designprozess
75 Seiten

D-96-01
Klaus Fischer, Darius Schier
Ein Multiagentenansatz zum Lösen von Fleet-Scheduling-Problemen
72 Seiten

1995

D-95-12
F. Baader, M. Buchheit, M. A. Jeusfeld, W. Nutt (Eds.)
Working Notes of the KI'95 Workshop:
KRDB-95 - Reasoning about Structured Objects:
Knowledge Representation Meets Databases
61 pages

D-95-11
Stephan Busemann, Iris Merget
Eine Untersuchung kommerzieller Terminverwaltungssoftware im Hinblick auf die Kopplung mit natürlichsprachlichen Systemen
32 Seiten

D-95-10
Volker Ehresmann
Integration ressourcen-orientierter Techniken in das wissensbasierte Konfigurierungssystem TOOCON
108 Seiten

D-95-09
Antonio Krüger
PROXIMA: Ein System zur Generierung graphischer Abstraktionen
120 Seiten

D-95-08
Technical Staff
DFKI Jahresbericht 1994
63 Seiten

Note: This document is no longer available in printed form.

D-95-07
Ottmar Lutzy
Morphic - Plus
Ein morphologisches Analyseprogramm für die deutsche Flexionsmorphologie und Komposita-Analyse
74 Seiten

D-95-06
Markus Steffens, Ansgar Bernardi
Integriertes Produktmodell für Behälter aus Faserverbundwerkstoffen
48 Seiten

D-95-05
Georg Schneider
Eine Werkbank zur Erzeugung von 3D-Illustrationen
157 Seiten

D-95-04
Victoria Hall
Integration von Sorten als ausgezeichnete taxonomische Prädikate in eine relational-funktionale Sprache
56 Seiten

D-95-03
Christoph Endres, Lars Klein, Markus Meyer
Implementierung und Erweiterung der Sprache $\mathcal{ALCP}$
110 Seiten

D-95-02
Andreas Butz
BETTY
Ein System zur Planung und Generierung informativer Animationssequenzen
95 Seiten

D-95-01
Susanne Biundo, Wolfgang Tank (Hrsg.)
PuK-95, Beiträge zum 9. Workshop „Planen und Konfigurieren“, Februar 1995
169 Seiten

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

1994

D-94-15
Stephan Oepen
German Nominal Syntax in HPSG
— On Syntactic Categories and Syntagmatic Relations —
80 pages

D-94-14
Hans-Ulrich Krieger, Ulrich Schäfer
TDL - A Type Description Language for HPSG, Part 2: User Guide.
72 pages

D-94-12
Arthur Sehn, Serge Autexier (Hrsg.)
Proceedings des Studentenprogramms der 18. Deutschen Jahrestagung für Künstliche Intelligenz KI-94
69 Seiten

D-94-11
F. Baader, M. Buchheit, M. A. Jeusfeld, W. Nutt (Eds.)
Working Notes of the KI'94 Workshop: KRDB'94 - Reasoning about Structured Objects: Knowledge Representation Meets Databases
65 pages

Note: This document is no longer available in printed form.

D-94-10
F. Baader, M. Lenzerini, W. Nutt, P. F. Patel-Schneider (Eds.)
Working Notes of the 1994 International Workshop on Description Logics
118 pages

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

D-94-09
Technical Staff
DFKI Wissenschaftlich-Technischer Jahresbericht 1993
145 Seiten

D-94-08
Harald Feibel
IGLOO 1.0 - Eine grafikunterstützte Beweisentwicklungsumgebung
58 Seiten

D-94-07
Claudia Wenzel, Rainer Hoch
Eine Übersicht über Information Retrieval (IR) und NLP-Verfahren zur Klassifikation von Texten
25 Seiten

Note: This document is no longer available in printed form.

D-94-06
Ulrich Buhrmann
Erstellung einer deklarativen Wissensbasis über recyclingrelevante Materialien
117 Seiten

D-94-04
Franz Schmalhofer, Ludger van Elst
Entwicklung von Expertensystemen: Prototypen, Tiefenmodellierung und kooperative Wissensevolution
22 Seiten

D-94-03
Franz Schmalhofer
Maschinelles Lernen: Eine kognitionswissenschaftliche Betrachtung
54 Seiten

Note: This document is no longer available in printed form.

D-94-02
Markus Steffens
Wissenserhebung und Analyse zum Entwicklungsprozeß eines Druckbehälters aus Faserverbundstoff
90 pages

D-94-01
Josua Boon (Ed.)
DFKI-Publications: The First Four Years
1990 - 1993
75 pages

“May I Speak Freely?” Between Templates and Free Choice in Natural Language Generation

D-99-01
Documen+

Workshop at the 23rd German Annual Conference for Artificial Intelligence (KI '99), Bonn
14.–15. September 1999

Tilman Becker, Stephan Busemann (eds.)