Report from Working Group 2: Lexicalization and Architecture

Group Members: Robert Dale, Wolfgang Finkler, Richard Kittredge, Nils Lenke, Günter Neumann, Conny Peters, Manfred Stede

1 Introduction

This report summarises the results of the discussions held in Working Group 2: The group discussions focussed around three reasonably independent topics, and we have organised the report to reflect this.

In Section 2, we consider the problem of how, given an application that requires to have text be generated, we might go about determining the contents of the conceptual base that should be constructed to facilitate this process.

In Section 3, we consider the notions of canned text and templates, and try to elaborate on the proper role of these notions in a text generation system.

In Section 4, we look at the problem of deciding how a dialog system should determine the size of its contributions to a dialog.

A distinction that surfaced at various points during the workshop is that between what one might think of as different approaches to work in natural language generation. These might be characterised as follows:

NLG-PSY: This is natural language generation as a means to exploring and modelling psycholinguistic theory.

NLG-Lx: This is natural language generation as a means to developing theories of language structure and function.

NLG-App: This is natural language generation as a means to building applications. Ideas from any one subfield of NLG may provide useful insights for the others, and many researchers would be unhappy about being classified as working in only one of the three subfields (or, indeed, may not even agree that the three subfields are distinct). However, it should be borne in mind that the discussions reported below fall in the first instance within the domain of NLG-App. This is particularly true of Sections 2 and 3, but perhaps less so of the material in Section 4.

2 Determining a Conceptual Base for an Application

2.1 Starting Points

We assume that the task of a generation system is to take some elements of an input representation, and to produce some text that somehow corresponds to those elements of the input representation. This characterisation is broad enough to cover a very wide range of cases; for example, all of the following count as possible input representations:

- Sense data: this might be an appropriate input representation if we have a vision system and we want to describe what it can see;
- Tabular information: this covers the kinds of data dealt with by the report generation systems described by Dick Kittredge and Tanya Korelsky as a basis for workshop discussions; and
- The knowledge representation of an expert system.

2.2 **Intermediate Representations: The Problem and Our Assumptions**

In each case the problem is this: there may not be a straightforward mapping from elements and structures in the input representation to words and syntactic constructs in the target natural language. In such situations, we can think about constructing an intermediate representation - which we will refer to here, with some caution, as a conceptual base - that makes the generation task easier.

We come to this problem with certain assumptions in mind. One might take the view — particularly if one is working in the mould of NLG-Psy or NLG-Lx — that the conceptual representation to use is the one that people use, or the one that is indicated by intuitions about the nature of language. This is not the view we take here. Without taking a stance on conceptual realism, our position here is that even if such conceptual representations do exist, they are likely to be far more sophisticated and fine-grained than our applications require; and, indeed, that the cost of implementing such rich conceptual models rules them out as inappropriate for our particular task.

Our view, then, is that we should develop theories of the meanings or concepts underlying particular words or phrases such that those theories are already assumed to be domain and application dependent, and that we should not expend the effort required to develop any kind of universal conceptualisations. Thus, for example, we might develop a micro-theory of the concept of "increase" in the domain of employment statistics domain, without expecting to find that the same micro-theory will be useful or usable in another domain.

2.3 A Principled Approach to Developing a Conceptual Base

On the basis of these assumptions and observations, we can then propose a methodology that can be applied to derive an appropriate conceptual base for a given application. The steps of the procedure are as follows.

1. Collect the words and phrases that appear in an appropriate set of sample texts: this is effectively a corpus analysis task, and can make use of whatever tools, such as

concordances, seem appropriate. The aim here is to identify the concepts that need to be represented in order to generate texts in the domain. Of course, intuition can be used to extrapolate from the available data to other plausible concepts that might need to be included: if our weather report corpus indicates that the concepts underlying the words "north", "west" and "east" will be useful, but does not contain any instances of the word "south", it seems reasonable to suppose that this is only the case for accidental reasons.

2. Identify the relevant semantic distinctions in the domain: the point here is that the set of lexical items and phrases we have identified may be larger than the set of concepts required to underly these words and phrases. Thus, for example, if the words "increase" and "rise" seem to mean the same thing, and the variation between their use .. looks as if it is best explained by local contextual or stylistic factors, then this argues for having only one concept of "increase" that can be realised by means of two distinct lexical items; this abstraction simplifies any reasoning processes we might need to carry out at the conceptual level. Since we are not committed to embodying a particular conceptual model in our system, we can provide a conceptual base on ly for the distinctions we need to have, and no others. The nature of the application will determine what concepts it is useful to have: if we are building a system which produces reports in a number of languages, then it is likely to be useful to have a level of representation which abstracts across the lexical differences in the languages.

3. Having developed a conceptual base in this way, we can then address the problem of providing a mapping from the input representation to this more generator-friendly conceptual representation. This can possibly involve re-designing the distinctions made in the previous step, as the mapping will often be non-trivial, and working on this task can inform the previous one.

Of course, there's nothing new about this methodology: indeed, it's very close to what people do when they build interlingua-based machine translation systems. However, we feel it is important to state the obvious, if only because the obvious is easy to lose sight of: it is all to easy to fall into abstract arguments about sophisticated representations being required in order to allow certain kinds of reasoning, but if one's application never needs to carry out the kind of reasoning in question, then the subtle distinctions in the representation are likely to be of no value to the application in hand.

After approaching several domains via this methodology, it may be appropriate to look for generalisations across domains, with the hope that some scope for reuse can be found; but one shouldn't expect too much here. It is always, of course, possible to build generalisations by abstracting further and further away from the real data at hand; but to do so may not be of any real benefit.

This viewpoint is possibly over-pessimistic: we might find, for example, that lexical semantic information about verbs may be transferable although the connection to the underlying conceptual base needs to be reworked for each domain. But our general point is that one should not assume from the outset that transfer will necessarily be easy.

Our principle, then, for determining an appropriate conceptual base for an application is this:

Don't introduce any more conceptual distinctions than you have to in order to meet the needs of the underlying application on the one hand, and to get the desired generation behaviour on the other.

This may sound rather trivial, but to the extent that this is true, it is unavoidable: if we really are only interested in building a working application, and are not playing at developing broad-coverage ontologies, then the particular set of conceptual distinctions we require will vary from one domain and application to the next.

3 The Appropriate Use of Canned Text

3.1 The Problem

An issue which was raised during the first few days of the workshop was the question of the advantages and disadvantages of using either canned text or templates in a generation system, in opposition to the use of a full-blown sentence planning mechanism. Our group decided to consider whether it was possible to determine principles that would help a system designer to decide on the correct balance between these different mechanisms.

3.2 Some Definitions

3.2.1 Templates and Canned Text

Here we found a problem: what exactly do we mean by the terms "canned text" and "template"? These terms are often used with the intended meaning being left to the reader's intuitions; pinning down precise definitions is less easy. We offer the following.

Template: A parameterised correspondence between input representations and output text, where the output text varies as the input parameters vary.

Canned Text: A template with no parameters.

These are made clearer by example. Suppose we have an input symbol of the form "greet", which should be realised always as the sequence of words "Good morning". This is an instance of canned text, or what we will call a "canned text item". As an alternative, suppose that our input form is parameterised to take an internal symbol that corresponds to the person to whom the greeting is addressed; then, given the input "greet(xl)" where xl is the internal symbol corresponding to the person named Noam Chomsky, the output might be "Good morning, Mr Chomsky", with appropriate changes to the name of the addressee as the parameter varies. This is then an instance of a "template item".

This, again, may sound rather trivial, but note that by these definitions, the items that populate the lexicons of many systems - where correspondences are drawn between semantic types and lexical items - are to be considered canned text items. In those cases where the lexical item is responsible for carrying out some morphological variation on the basis of some input parameter (such as tense or number, for example), then we have template items.

Of course, we don't typically think of such small elements as canned text or templates: there's an intuition that there ought to be more than one word, at least, for something to qualify as canned text or template. But this is misleading, since (a) some "lexical items" may indeed contain more than one word (consider phrasal verbs); and (b) the process by means of which our "greet(x)" template item produces variable output is formally no different from a lexical entry that allows us to produce either of the words "child" or "children".

One way of subcategorising these mechanisms would be to note that structures like the "greet(x)" template item combine the canned phrase with any arbitrary string, whereas the morphological derivation of a word complete with either singular or plural marking is carried out on the basis of a parameter which has a finite number of values.

It should also be noted that, given our definitions above, any entire generation system can be characterised as a template: it takes different inputs and produces different outputs. If we are to make sensible and precise use of the notions of template and canned text, it looks like we need to do a little more work to delimit different kinds of transductions.

3.2.2 **Different Kinds of Canned Text**

We did not address further the question of how one might carve up this space of possible transduction devices; however, we did find it useful to elaborate a little further on the different kinds of canned text we might want to make use of.

When we think of canned text, we typically think of ASCII strings whose internal structure is not available. However, this view conflates two aspects of cannedness. A piece of text can, indeed, be canned in the sense that we do not have access to its internal structure; so, for example, we might have a dialog system which includes in its repertoire of canned text items a mapping from some internal symbol to the ASCII string "the red switch in the corner of the display". In such a system, if the user asks a question like "Why did you describe the switch as being red?", the system has little hope of answering the question: quite apart from the unavailability of information that would allow the system to reason about its own reasoning here (note the parallel with discussions of the need for keeping track of intentions in discourse planning), the system does not even have a clue about the internal structure of the noun phrase it has just generated.

But this lack of structure is only one aspect of cannedness. The other facet of cannedness is that, because the text is canned, we do not have to carry out the work of building the text each time: we just look up the internal symbol in some table, and retrieve the canned text that corresponds to that symbol. But notice that this "precompilation" aspect of cannedness does not necessitate the absence of structure: we could just as easily return a fully-formed noun phrase structure from our table, perhaps even annotated with intentional information when this is invariant.

Because of this, we find it useful to distinguish two kinds of canned text:

Canned strings: A canned string has no internal structure, and is simply a sequence of ASCII characters (typically, but not necessarily, corresponding to more than one word in the natural language).

Canned structures: A canned structure has internal structure, but has been prebuilt so that this structure does not have to be rebuilt on each occasion of use. The specific contents of this structure can vary: an obvious type of content is syntactic structure, but one can imagine many other kinds of structure that could be build in, including semantic or intentional information, and information such as might be required for generating hypertext links.

In addition to this distinction, we might also attach annotations to a canned text item that determine the appropriate contexts for its use.

All of the above has been expressed in terms of canned text, but of course the same observations apply to templates, so that we can think of string templates and structured templates. A simple example of a structured template would be a phrasal lexical item for an idiom like "kick the bucket", where the template would allow the tense marking on "kick" to be specified by parameter while at the same time providing the sequence of words in the form of a fully analysed verb phrase.

3.3 A Principle for Deciding on the Mix

So much for our hunt for useful definitions of these terms. There remains the question: how does a system designer decide when to use canned text, when to use templates, and when to use full-blown sentence planning techniques? Our principle here is in the same minimalist mould as that which we offered in the case of developing conceptual representations:

> Adopt a mixed approach that yields maximum efficiency and/or elegance in terms of design and execution, depending entirely on the purpose of the system.

The point is, again, the same as in the case of our previous principle: there are no hard and fast rules here, since it all depends on the particular application. The component of a programming language compiler that generates error messages uses canned text, or in some circumstances, templates (in order to be able to include line numbers, for example); it would be quite absurd to generate these messages from first principles using a sentence planner. However, at some point in any application, a threshold is reached where the designer deems it more useful to abstract across some set of circumstances and deal with these by means of a paramterised procedure or rule, rather than on a case by case basis. This threshold is determined by many application-specific characteristics. If what we are interested in is building systems, rather than building a model of language, then principles such as efficiency, economy, and elegance of system design dictate the correct mix.

4 Chunking Text in Dialog

This section of our report differs from the foregoing two in that no "minimalistic" approach was used. Instead, we tried to investigate the space of possible parameters and solutions up to a certain depth. As a result, we can only give more questions than (very tentative) answers.

The topic of the following is: Imagine you have to provide a "large" contribution in a dialogue, e.g.. a longish route description. How should you organize it into chunks of information, e.g. divide it into several smaller contributions? Which structure should be imposed on that, and how could this structure be signalled to the user?

Of course, to a certain degree this can also be applied to small contributions.

4.1 How can the size of a contribution be measured?

You have to differentiate between its size on the conceptual level ("size of content") and on the surface level ("size of string").

Size of string is measured easiest but there is no 1:1-correspondence between size of content and size of string. The ratio of "informational unit per word" ("density") differs between dense language, e.g. in a scientific talk, and shallow language, e.g. in a politicians" talk :-). String size also depends on the expressibility of concepts in a certain language, which depends on the user model. So, it is difficult to predict the string size from the content size.

This could indicate a problem for serial generation architectures!

How can the size of content be measured? Number of propositions? But then, what is a proposition?

4.2 Which factors that impact on the size of a contribution can be identified?

Here are some:

- The amount of noise in the channel.
  If there is plenty of noise, it would be a wise idea to keep contributions small in order to give the hearer a chance of signalling trouble regularly and to keep repetitions in case of trouble small. On the other hand, Noise could increase the need of redundancy in the contributions, that is, increase their string size.

- The genre.
  In small talk, contributions should not be too large in order to give the other participants a chance. In Interviews it is quite O.K. for the interviewed to keep the floor most of the time.

- The predictability of the information (given vs. new).
  In normal speech, rest of words or sentences can be guessed if the beginning was understood. If predictability is lowered, contributions should become smaller. An extreme example: Giving names via the telephone. Size of chunks (= contributions) can be as small as one letter:

```
A: B: A: B: D
A: B: A:    yes
            a
            yes
            l
            yes
            e
```

Certain rhetorical "rules" and schemata, e.g. making lists consist of THREE items rather than two ore four.

4.3 How can the informational structure of contributions be organized?

All texts have got some kind of (hierarchical) informational structure. This best be seen in argumentative texts, where it is the argument structure/tree. This structure has to be grasped by the reader/ hearer. The following points can be made about that:

- Structure can be signalled by "cue words/phrases", e.g. "first", "coming back to" and the like. Intonation and different voice levels can be used, too. "Advance organizers" can give hints to the hearer about the organization of the following contribution(s).

- In speech, the structure has to be somewhat "flatter" than in written texts, because of the limited processing capacity of participants (memory etc.) and due to the linear form of speech.
- This might lead to a loss of information (which was encoded in the original structure) if the tree is flattened for the purpose of the dialogue.
- The necessary degree of flattening might depend on your audience. (Computer scientists who are used to tree structures vs. your granny)
- In dialogues via computer terminals, more techniques for signalling structure are available, e.g. itemization and indentation.

4.4 Dialogue aspects.

Don't forget that in a dialogue there is a hearer that might be able to help you.

- So, after each contribution s/he should get a chance of signalling success or failure. To eliciting these "back channels" you can use pauses, tag questions and intonation.
- If the hearer interrupts you unexpectedly this might be a valuable hint for chunking: At least this chunk/contribution was too long! Your program should have methods to deal with this. •3

5 Conclusions

In conclusion, we would return to our initial statements, and encourage generation researchers to make clear, when they address questions of principle, which perspective on generation they are taking. It is our view that this will lead to much less confusion and disagreement in the field than has been evidenced so far.