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 be-

cause 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.