Kalos – A System for Natural Language Generation with Revision

Ben E. Cline and J. Terry Nutter
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061
benjy@benjy.cc.vt.edu/jtn@vtopus.cs.vt.edu

Abstract

Using revision to produce extended natural language text through a series of drafts provides three significant advantages over a traditional natural language generation system. First, it reduces complexity through task decomposition. Second, it promotes text polishing techniques that benefit from the ability to examine generated text in the context of the underlying knowledge from which it was generated. Third, it provides a mechanism for the interaction of conceptual and stylistic decisions. Kalos is a natural language generation system that produces advanced draft quality text for a microprocessor users' guide from a knowledge base describing the microprocessor. It uses revision iteratively to polish its initial generation. The system performs both conceptual and stylistic revisions. Example output of the system, showing both types of revision, is presented and discussed.

Introduction

Natural language connected text systems produce multiple sentence texts, from one to several paragraphs long, to satisfy a particular discourse goal. The system must select and order concepts from a potentially huge knowledge base and translate them into cohesive surface text. The limited capabilities of state-of-the-art connected text generation systems attests to the difficulty of implementing robust, general systems in this area. Two problems at the root of this difficulty are the lack of robust generation techniques and the complexity of the generation task.

Connected text generation systems currently function in limited domains and for limited discourse goals, do not produce formal, polished text, and do not generalize well either to new domains or to new types of discourse.

A connected text generation system must both select and order concepts from its domain knowledge base to satisfy a given discourse goal (e.g. describing some object), and convert these concepts into a cohesive surface text. This process is complicated by a number of factors. The system must be discriminating in what it says, neither stating obvious facts nor omitting salient ones. The facts must be ordered logically, and the text must be cohesive. Current generation systems generally achieve these goals by specializing their techniques either to features of the task (limited discourse goals) or to features of the domain.

The lack of robust techniques results to a large extent from the inherent complexity of the task. The complexity has other effects as well. Most current systems attempt to deal with generation by decomposing the task into one or two stages, usually by isolating the process of selecting and ordering concepts to meet the discourse goal from that of converting the concepts into natural language. But this decomposition limits as well as simplifies.

The system reported here uses revision in a knowledge intensive environment to improve on generation techniques and to deal with generation complexity. Our revision techniques address both conceptual and stylistic defects in draft text. The Kalos natural language generation system was developed to demonstrate these revision techniques. It is a complete generation system that generates portions of a draft users' guide for a microprocessor.

Why Revision?

Hays and Flower (1980) developed a model for human text production in which revision reduces strain on human authors by reducing the number of decisions that must be dealt with during any part of the task. Vaughan and McDonald (1986), Yazdani (1987), Cline (1991), and Meeter (1991) have suggested that text revision may likewise aid natural language generation. Vaughan and McDonald (1986) and Meeter (1991) focused on stylistic revisions only; Yazdani (1987) and Cline (1991) suggest that conceptual revisions are also useful.

Revision provides three benefits for natural language generation systems:

- It reduces system complexity through task decomposition and modularity.
- It provides an architecture for text polishing techniques that benefit from the ability to examine gen-

erated text in the context of the underlying structures from which it was generated.

- It allows interaction between conceptual and stylistic decisions.

Natural language generation is a formidable task: reducing complexity is crucial. Using a revision architecture simplifies generation module design by postponing many decisions to the revision module. As Yazdani (1987) points out, this type of architecture is common in the construction of complex software systems such as compilers.

There is another software engineering benefit to incorporating a revision module into a natural language generation system. The revision module is a natural place to isolate domain-specific linguistic knowledge and knowledge that relates to both surface and deep generation modules, thus producing a more robust, maintainable, and adaptable generation system.

The revision model also promotes text polishing techniques that benefit from the ability to examine generated text in the context of the underlying structures from which it was generated. For instance, a revision component is the ideal place to identify and eliminate ambiguities in the generated text. An initial generation module could try to avoid generating ambiguous text, but the complexities involved are overwhelming. Once the text has been generated, reading it to locate ambiguities is a less demanding. Hence it makes sense to locate and eliminate ambiguities in a revision module, which has access to both the surface text and information about its origin.

Some problems related to word sound, such as repetition and rhyming, are best dealt with by a revision module. For example, consider a knowledge base with two concepts *register* and *data register*, that are related as superclass and subclass. A natural language generation system might produce, "The D0 data register is a register" (structurally analogous to "The F-150 pickup is a light-duty truck"). The sentence is awkward because of the repeated "register." Such surface-level problems can be dealt with during initial surface generation, but again, doing so complicates the generator.

A revision architecture also addresses the problem of lack of interaction between conceptual and surface decisions (McKeown & Swartout 1987). Traditional systems make conceptual decisions first, and then generate surface text. This architecture does not allow lexical choices to influence conceptual decisions. A revision architecture lets the system change conceptual decisions to facilitate using particular words or phrases. For example, preferring the term *address space* to address bus size in describing a microprocessor affects the organization of the text. Consider a description of the address bus of the Zilog Z-80 microprocessor that refers to address bus size (an attribute of the address bus):

- The address bus of the Zilog Z-80 microprocessor is sixteen bits wide. (1)

The same fact can be rephrased to refer to the address space:

- The Zilog Z-80 has a sixty-four kilobyte address space. (2)

Both sentences reflect the same information. But the first sentence relates it as an attribute of the address bus, while the second sentence makes a statement directly about the processor. The second sentence both uses a preferred way of describing the processor's maximum memory size and gives an important feature of the microprocessor. It is thus desirable to include it in an overview paragraph of the microprocessor rather than in a following paragraph describing its buses. The apparent surface preference for one descriptive term over another thus affects the deep structure of the text to be generated.

Types of Revision

Revision, whether by humans or computers, takes two forms (Cline 1991). *Stylistic revision* occurs when the surface text is changed without altering the meaning of the text or the order of concepts. *Conceptual revision* occurs when the meaning of the text or the order of concepts changes. Replacing a noun phrase with a pronoun and compounding sentences are examples of stylistic revisions. Reordering, adding, or deleting text results in a conceptual revision. Examples of conceptual revisions are adding an example to existing text and reordering attributes of an object being described so that quantifiers are given first.

An example of a conceptual revision was given in the previous section (sentences 1 and 2). Consider the following example of stylistic revision. Sentences (3) and (4) are draft text:

- D0 is a register. (3)
- D0 is 32-bits wide. (4)

A simple stylistic revision is to render sentence (3) as the compound noun "the D0 register" and to use it as the subject of sentence (4), producing

- The D0 register is 32-bits wide.

Although revision-based systems have been proposed for some time, only limited systems have been produced so far. The most advanced systems incorporating revision are the weiveR system (Inui, Tokunaga, & Tanaka 1992) and the STREAK system (Robin 1993). The weiveR system is limited to stylistic revision of Japanese text. It focuses on repairing structural ambiguity and sentence complexity problems such as those associated with sentence length and depth of embedding. weiveR currently does not perform deep generation or conceptual revision, although the authors of that system feel that revision should be more broadly applied for most of the reasons discussed above.

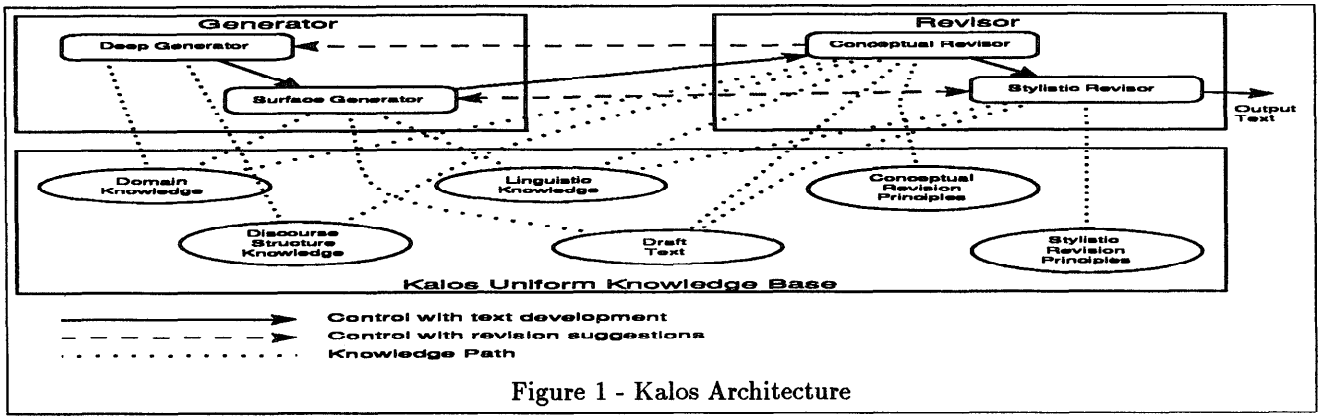

Figure 1 - Kalos Architecture

The STREAK system performs revision to add historical information to draft text. This system produces draft text from wire reports and then adds historical information at the word and phrasal level to help explain the significance of the information in the wire report. It considers both conceptual and stylistic concerns when making revisions. Although the architecture of STREAK is interesting, the system reported in Robin (1993) is incomplete and performs only one type of revision on a single sentence.

Revision Architecture

Kalos consists of a uniform knowledge base and two main modules: a generation module with deep and surface generation submodules, and a revision module consisting of conceptual and stylistic revision submodules (see figure 1). The deep generator selects and orders concepts to meet some discourse goal. The surface generator converts the concepts selected by the deep generator into surface text. The conceptual revisor makes suggestions for improving conceptual defects in the text, while the stylistic generator makes suggestions for improving stylistic defects in the text.

Unlike those of a traditional generation system, Kalos's deep and surface generators are relatively simple. Initially, they produce simple draft text. Many of the decisions made by traditional generators are postponed for consideration by the revision modules. After generating the initial text, Kalos improves it iteratively in two cycles. The first cycle consists of the deep generator, surface generator, and conceptual revisor. In each iteration, the deep and surface generators produce text and the conceptual revisor examines it for defects. If the revisor finds defects, it produces suggestions to improve the text. The deep generator uses the suggestions to regenerate the text. Revisions are cumulative, i.e., no revision suggestion is ever retracted in a later pass. The cycle ends when the conceptual revisor finds no further defects.

Neither the decision not to retract revisions nor the termination condition is fundamental to the model. Both were chosen for simplicity. Other systems, for example, could use a measure of text quality to determine when to stop the revision process.

The stylistic revisor cycle begins after all conceptual revisions are complete. This cycle begins with the stylistic revisor, which reviews the draft text and produces revision suggestions to improve it. These suggestions go to the surface generator, which uses them to regenerate the text. This cycle ends, and the final text is output, when no more revision suggestions can be applied.

We use two principles in deciding how to decompose the generation task. The first is based on locality of decision making. The generators make decisions based on local information while the revisors make decisions based on a wider set of knowledge. The deep generator focuses on selecting single concepts at a time while the conceptual revisor considers the interaction between selected concepts such as whether a concept adds redundant information. The surface generator focuses on generating a single, simple sentence for each concept to be surfaced. The stylistic revisor makes decisions that involve more than one sentence, such as combining two sentences into a compound sentence.

The second decomposition principle is based on the knowledge sources needed for a task. To keep the generators simple, they make their decisions based on traditional knowledge sources. If additional knowledge is needed to make some decisions, these decisions are made in one of the revisors. For example, the deep generator traditionally doesn't consider linguistic knowledge, but some conceptual decisions, such as the removal of sentences that are redundant due to surface-level effects, require linguistic knowledge. These types of decisions are postponed for the conceptual revisor where the addition of linguistic knowledge has less impact on the computational needs of the module.

We believe that the best architecture for a revision system uses a uniform knowledge base containing all the system knowledge similar to the knowledge base described in Cline and Nutter (1992, 1994a). This approach lets revision modules determine the intent of each part of the generated text quickly, and determine what alternative choices are available for regenerating

the text. Only with access to the full system knowledge can revision components execute intelligently and efficiently. We encode all the knowledge in a uniform representation to which a single inference technique can be applied. In Kalos, all knowledge is encoded as SNePS-2.1 (Shapiro 1992) semantic networks, and SNePS inferencing is the only inference technique used. Contrast this to a traditional natural language generation where the surface generator knowledge, encoded as an Augmented Transition Network (ATN) or other form of grammar rules, is hidden from the other stages of generation.

The use of a uniform knowledge base places a computational burden on the system due to the size of the knowledge base. To address this, Kalos dynamically partitions the uniform knowledge base to meet the requirements of each generation and revision task, using path-based techniques and pattern matching over inferencing where possible, and using conceptually restricted knowledge bases (Cline 1994b).

Kalos

Kalos was developed as a testbed for revision techniques. The system produces advanced draft text for portions of a microprocessor users' guide from a domain knowledge base describing a particular microprocessor. Because of the limitations in state-of-the-art natural language generation techniques, we have decided to concentrate on the generation of draft quality text.

The Kalos deep generator is a scriptal-based generator (Hovy 1988) that uses discourse schemata to select and order concepts to describe a microprocessor. The schema slot-filling mechanism uses SNePS-2.1 pattern matching and inferencing rules. The surface generator is based on a type of unification grammar (Kay 1984) encoded into SNePS-2.1 semantic networks so that the revisors can inspect the grammar rules.

The Kalos conceptual revisor currently performs four types of conceptual revisions:

- Removal of redundant and superfluous information
- Application of domain-specific preferred words and phrases with conceptual effect
- Proper ordering of attributes
- Handling of inordinately long lists

The conceptual revisor examines the generated text and the underlying structures from which it was generated for defects. Defects occur because the deep generator is relatively simple. Many decisions are postponed for consideration by the conceptual revisor, thus reducing individual module complexity through task decomposition. The revisor can suggest that a schema slot be removed, that an unfilled schema slot be filled, that a different schema slot choice be selected, and that a list of attributes be reordered.

Conceptual revisions rarely conflict, because of the design of the schema templates and slot-filling salience rules. Hence Kalos uses a simple priority scheme to deal with revision suggestions that affect the same schema slot, in which requests for deletion take precedence over other suggestions. Conceptual revision suggestions are never retracted.

The Kalos stylistic revisor performs the following stylistic revisions:

- Suggest use of anaphora
- Suggest sentence and phrase compounding
- Suggest the use of preferred words and phrases with stylistic effect
- Suggest thematic progression constructs
- Suggest other cohesive constructions

Because the surface generator initially generates each schema slot as a single simple sentence, the stylistic revisor must combine the sentences into a cohesive whole. It examines two consecutive sentences at a time, perhaps making a number of suggestions to improve the sentences. After all the sentences have been analyzed, the suggestions are analyzed for conflicts. Revision suggestions are weighted to indicate their desirability and amount of structural change. In the case of conflicting suggestions, those with greater weights are favored other those with smaller ones. After conflicts have been resolved, the surface generator receives the remaining suggestions and regenerates the text. Later cycles may implement suggestions removed in earlier ones, if they still apply after the more extensive changes have been made.

Most of the stylistic revision types are straightforward; we restrict our discussion to thematic progression (Glatt 1982). Two sentences are in thematic progression order if they both have the same subject or if the rheme (what is said) of the first sentence is the theme (what is talked about) of the second. Sentences in thematic progression order tend to be easier to follow than those that are not.

For example, consider the two sentences:

- The M68000 supports memory-mapped I/O.
- There are 9 M68000 address registers. (5)

The second sentence can be restated using M68000 as the subject:

- The M68000 supports memory-mapped I/O.
- The M68000 has 9 address registers. (6)

Further cycles replace the subject of the second sentence by a pronoun, etc.

Kalos Example

A complete description of all kinds of revision that Kalos performs is beyond the scope of this paper. To illustrate system performance, we discuss the generation and revision of the first introductory paragraph of the Motorola M68000 microprocessor. This example illustrates how revision can be used to generate draft level text from relatively simple generators.

```
((the M68000 is a microprocessor)
 (the M68000 supports memory-mapped I/O)
 (the M68000 address bus is an address bus)
 (the M68000 address bus is 24 bits wide)
 (the M68000 data bus is a  data bus)
 (the M68000 data bus is 16 bits wide)
 (the M68000 address registers are address registers)
 (There are 9 M68000 address registers)
 (the M68000 address registers are 32 bits wide)
 (the M68000 data registers are data registers)
 (There are 8 M68000 data registers)
 (the M68000 data registers are 32 bits wide)
 (the M68000 instructions are instructions)
 (There are 82 M68000 instructions))
```

Figure 2 - Sample Kalos Initial Generation

During initial generation, Kalos selects a schema and generates each slot as a simple sentence (figure 2). After two passes by the conceptual revisor, the text of figure 3 is produced. The first revision describes the M68000 as a "16-bit microprocessor" instead of just a "microprocessor," the natural category to which it belongs (Cline and Nutter 1990). This revision occurs because "8-bit microprocessor," "16-bit microprocessor," "32-bit microprocessor," and "64-bit microprocessor" are domain-specific preferred terms. Using knowledge about the relationship of data bus sizes and these terms, the conceptual revisor infers that the M68000 is a member of subordinate class *16-bit microprocessor* and causes the deep generator to fill the schema slot giving taxonomic information about the M68000 with this deduced concept instead of the frame indicating that the M68000 is a member of the *microprocessor* class.

The second conceptual revision results from the domain-specific preferred term "address space size." The conceptual revisor deduces the address space size from the address bus size and adds this concept as an attribute of the microprocessor. To make this revision, the conceptual revisor inspects the linguistic knowledge base to determine what type of knowledge base frame will trigger this term. It then tries to deduce this type of frame for any object of the type associated with the term. If it can infer such a concept, it instructs the deep generator to include it in the instantiated schema it produces.

Both preferred phrase revisions result in redundant information. When the M68000 is described as a "16-bit microprocessor," the size of the data bus becomes redundant. Similarly, the size of the address bus becomes redundant after the address space size is given. The second pass through the conceptual revisor detects the redundancy and removes the descriptions of the data and address buses. (The buses are still described in detail in a following paragraph which is not shown in the example.)

The conceptual revisor also removes inherently redundant sentences like "the M68000 data registers are

```
((the M68000 is a 16-bit microprocessor)
(the M68000 has an address space size of 16 megabytes)
(the M68000 supports memory-mapped I/O)
(There are 9 M68000 address registers)
(the M68000 address registers are 32 bits wide)
(There are 8 M68000 data registers)
(the M68000 data registers are 32 bits wide)
(There are 82 M68000 instructions))
```

Figure 3 - Kalos Output After Conceptual Revision

```
((the 16-bit M68000 microprocessor has an address space
  size of 16 megabytes and supports memory-mapped I/O)
 (it has 9 32-bit address registers and 8 32-bit data
  registers)
 (it executes 82 instructions))
```

Figure 4 - Kalos Output After Stylistic Revision

data registers." The lexicon entry for the subclass *M68000 data register* contains the fact that its members by definition belong to the class *data-register*, indicating that in this case, stating this class membership is redundant. The conceptual revisor examines the lexicon and domain knowledge base to determine that sentences of this kind should be deleted.

After the first pass through the conceptual revisor, two attributes of the M68000 are listed in the following order:

- The M68000 supports memory-mapped I/O.
- The M68000 has an address space size of 16 megabytes.

During the second pass of the conceptual revisor, these two attributes are reversed to list the quantitative attribute before the qualitative one.

Figure 4 shows the text after stylistic revision. The first stylistic revision pass combines the first two sentences by converting the first into a noun phrase and using it as the subject of the second sentence, producing:

- The M68000 16-bit microprocessor has an address space size of 16 megabytes.

The next revision pass splits the term "16-bit microprocessor" to let the quantitative descriptor be listed before "M68000," and the sentence is compounded with the next sentence.

The sentences giving the number of address and data registers are modified to have the same subject as the previous sentence, maintaining thematic progression order. An example of this revision is shown by sentences (5) and (6) in the last section. On the next revision pass, the sentences listing the number of registers are combined with sentences listing register sizes.

On the final stylistic revision pass, Kalos uses pronouns as appropriate.

Kalos currently generates the two opening paragraphs for a description of the Motorola M68000 mi-

croprocessor and several smaller texts. The two paragraphs describing the M68000 contain 49 sentences after initial generation. After conceptual revision, the text contains 29 sentences. The final text contains 14 sentences (many of which are compound sentences). To extend the text generated by Kalos, additional knowledge about the M68000 could be added to the system.

Conclusion

Kalos illustrates the potential of revision-based natural language generation in a knowledge intensive environment. Revision reduces module complexity by increasing task decomposition and modularity. It also provides a welcoming architecture for implementing a number of useful techniques that benefit from examining generated text in the context of the underlying structures from which it was generated. Finally, revision enhances the interaction of conceptual and stylistic decisions.

A number of questions remain to be examined. One relates to the way Kalos performs revisions: revisions are never retracted and conceptual revision precedes stylistic revision. Relaxing these two constraints would allow greater flexibility, but would increase the complexity of the system. Termination criteria, conceptual and stylistic revision interaction, and revision ranking techniques would need to be studied.

Another major enhancement to Kalos would be the addition of ambiguity checking. The stylistic revisor is an ideal place to check for ambiguity because it has access to both the surface text and the underlying knowledge from which the sentence was generated. With an appropriate natural language understanding module, the surface text could be read for ambiguities. From the underlying structures of the surface text, Kalos could determine the intent of the text and look for other ways to generate it. If no unambiguous text could be found, Kalos could indicate that fact so that a human author could polish the text.

References

Cline, B. E. & Nutter, J. T. 1990. Implications of Natural Categories for Natural Language Generation. In *Current Trends in SNePS - Semantic Network Processing System,* D. Kumar, ed., Springer-Verlag, Berlin, 153-162.

Cline, B. E. 1991. Conceptual Revision for Natural Language Generation. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics,* 347-348. Berkeley, CA: Association for Computational Linguistics.

Cline, B. E. & Nutter, J. T. 1992. Knowledge-Based Natural Language Generation with Revision. In *Proceedings of the 5th Florida Artificial Intelligence Research Symposium,* 223-227. Ft. Lauderdale, Florida: Florida Artificial Intelligence Research Symposium.

Cline, B.E. & Nutter, J.T. 1994a. Generating and Revising Text: A Fully Knowledge-Based Approach. *International Journal of Expert Systems, Research and Applications,* 7(2).

Cline, B. 1994b. *Knowledge Intensive Natural Language Generation with Revision.* Ph. D. Dissertation. Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA.

Glatt, B. S. 1982. Defining Thematic Progressions and Their Relationship to Reader Comprehension. In *What Writers Know: The Language, Process, and Structure of Written Discourse.* Nystrand, M, ed., New York: Academic Press.

Hayes, J. R. & Flower, L. S. 1980. Identifying the Organization of Writing Processes. In *Cognitive Processes in Writing,* L. W. Gregg and E. R. Steinberg, eds., Lawrence Erlbaum, Hillsdale, NJ, 3-30.

Hovy, Eduard H. 1988. *Generating Natural Language Under Pragmatic Constraints.* Hillsdale, N. J.: Lawrence Erlbaum Associates.

Inui, K., Tokunaga, T., and Tanaka, H. 1992. Text Revision: a Model and Its Implementation. In *Aspects of Automated Natural Language Generation,* R. Dale, E. Hovy, D. Rosner, and O. Stock, eds., Springer-Verlag, Berlin, 45-56.

Kay, M. 1984. Functional Unification Grammar: a Formalism for Machine Translation. *Proceedings of the Tenth International Conference on Computational Linguistics.* Stanford, California.

McKeown, K. R. & Swartout, W. R. 1987. Language Generation and Explanation. *Annual Review of Computer Science.* Volume 2.

Meteer, M. 1991. The Implications of Revision for Natural Language Generation. In *Natural Language Generation in Artificial Intelligence and Computational Linguistics,* C. Paris, W. Swartout, and W. Mann, eds., Kluwer Academic Publishers, Boston.

Robin, J. 1993. A Revision-Based Generation Architecture for Reporting Facts in Their Historical Context. In *New Concepts in Natural Language Generation: Planning, Realization and Systems,* H. Horacek and M. Zock, eds., Printer Publishers, London, 238-268.

Shapiro, S. & The SNePS Implementation Group 1992. *SNePS-2.1 User's Manual.* Department of Computer Science, State University of New York at Buffalo, Buffalo, NY.

Vaughan, M. M. & McDonald, D. D. 1986. A Model of Revision in Natural Language Generation. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics,* 90-96. New York, NY; Association for Computational Linguistics.

Yazdani, M. 1987. Reviewing as a Component of the Text Generation Process. In *Natural Language Generation,* G. Kempen, ed., Martinus Nijhoff Publishers, Dordrecht, 183-190.