MODELING EXTEMPORANEOUS ELABORATION

Marie A. Bienkowski
Bell Communications Research
445 South Street, MRE 2P358
Morristown, NJ 07960-1961, USA

ABSTRACT

Intelligent problem solving systems must be able to express their results in a coherent and flexible manner. One way this can be done is by *extemporaneous elaboration*, the method of language production that underlies more skilled tasks such as explanation. This paper outlines a computational model for extemporaneous elaboration that is implemented in a computer model called Extemper, shows examples of its operation, and compares it with other models of language production. Extemper contains the four components minimally required for elaboration: 1) an efficient method for linearizing a knowledge structure, 2) a translation/selection mechanism for producing a conceptual textbase from the knowledge structure, 3) local coherence operators which provide local connections between textbase elements, and 4) a conceptual generator to translate the coherent textbase into English.

Introduction

An intelligent problem solving system may be required to participate in a purposive discourse with a user. Purposive discourse is dialogue where a well-defined goal is pursued in a stereotyped and efficient - but not inflexible - way. Many common types of purposive discourse that could occur with an intelligent system, such as tutoring, explaining, and advising, require lengthy elaborations. Automated reasoners or problem solvers, in particular, are often required to explain, describe, or justify, that is, to *elaborate on*, their solutions. The process of producing unrehearsed and unedited expositions during purposive discourse is called *extemporaneous* or *spontaneous* elaboration.

This paper describes a computational model for extemporaneous elaboration implemented in a computer model called Extemper. The model consists of four components. The first is a *linearizer* which provides the overall structure of the elaboration by either a) following a trace of the processing of a problem solver (*rehashing*) or, b) directing a *rehearsal* of the knowledge used by the system (i.e., instantiating the knowledge structures normally used for problem solving for use in language production). The second component is a set of *selectors* which determine what will be expressed based on considering the type of discourse, the discourse goals of the listener, and the relevance of domain-specific items.

The linearizer and selectors produce the overall conceptual form of the elaboration. It is given a coherent linguistic form by the combination of *local coherence operators*, which examine conceptual forms to track focus, add connective words, etc., and a *sentence generator* (Cullingford, 1986) which renders it in English. Extemper's core model of elaboration, then, consists of four components: a linearizer, selectors, local coherence operators, and a sentence generator. These constitute a minimal cognitive architecture for elaboration: *minimal* because more may be needed for other elaboration types (e.g., a component to build and maintain a listener model for tutoring), and *cognitive* because components like these have been proposed by psychologists for models of language production (e.g., vanDijk and Kintsch, 1983).

In the next section, I discuss how Extemper's components cooperate to produce an elaboration. In Section 3, I describe the type of domain that Extemper's minimal architecture is sufficient for: descriptions of the operations of a problem solver. That section also distinguishes rehearsal from rehashing; these are different ways of obtaining the knowledge to be elaborated. Before concluding, I compare Extemper with other systems designed for text and discourse production.

Extemper: A Computational Model

Extemper implements a four component, minimal cognitive architecture for elaboration to produce descriptions of the behavior of problem solving systems. The problem solving systems it interacts with are a route planner and an academic counseling system.[1] Extemper's linearizer, in cooperation with one of these problem solvers, provides a succession of pieces of the knowledge source being elaborated to *translators*. These meaning-preserving translators produce conceptual forms which are linked into a conceptual knowledge base and then processed by the selectors. The output of both the translators and selectors is represented using a version of the *E*clectic *R*epresentations for *K*nowledge *S*tructures (ERKS) frame representation system developed by Cullingford (1986).

The concept selection cycle builds a single ERKS meaning structure until meaning structure (ms) functions determine that a newly selected concept does not fit with the current one. The ms functions use a heuristic evaluation of the connectedness of propositions to determine when to generate a sentence. They rely on the semantic classes of concepts and the goal structure represented in the conceptual knowledge base. For example, concepts representing simple actions taken by the route planner are considered to be connected, and can be expressed in a single sentence if they occur consecutively. Actions that contribute directly to the same goal are also considered to be connected. Note that there may also be syntactic constraints on what may fit into one sentence (e.g., Derr and McKeown, 1984, describe how considerations of focus may force complex sentences to be generated) but those constraints would not apply at this level, since the syntactic form the meaning structure will take is not yet known.

After concept selection, the local coherence operators mediate between the conceptual form of the elaboration and its syntactic form by annotating meaning structures with instructions for the sentence generator. These instructions increase the connectedness or local coherence of the sentences to be generated; given and new information, for example, is computed at this level. The annotated meaning structures are stored in a textbase whose elements are translated into English by the sentence generator. The sentence generator operates by repeatedly looking up words to span conceptual forms and ordering the words and those concepts that were not spanned according to predicates stored with the words. Additionally, there are conceptual "sketchifiers" that alter or delete information to express a concept more succinctly.

Two important contributions can be found in this computational model for language production (in addition to its description of the minimal components needed for

[1]Extemper and the problem solving systems are implemented in Franz Lisp on a Pyramid 90x.

elaboration). One contribution is illustrated by Extemper's use of the knowledge embodied in an intelligent system for guiding an elaboration. Extemper follows the principle that if a natural ordering exists for a body of knowledge, it should be used to guide extemporaneous elaboration. A natural ordering is one that corresponds to some common sense use of the knowledge, such as a problem solver might exhibit.

Another contribution of this elaboration model is that it demonstrates that a language production mechanism can be designed to operate primarily on conceptual, language-free forms. Manipulations are performed on the conceptual form of an elaboration, not the linguistic form. This is possible in Extemper because information from the problem solver is converted into conceptual form by the meaning-preserving translators, and the selectors, local coherence operators, and sentence generator operate only on these forms. Use of a conceptual representation throughout the elaboration process makes possible a blurring of the distinction (traditionally made in language production systems) between "what to say" and "how to say it."

Linearization and Translation

When a topic is chosen for elaboration, the concepts to be communicated must be selected in an appropriate order. Extemper's linearizer is responsible for presenting pieces of a knowledge structure for selection in an order that is derived in some fashion from information inherent in the structure itself. To gain access to this ordering information, the linearizer must cooperate closely with the intelligent system that is producing or manipulating the knowledge. I discuss this further in Section 3 while describing the problem solving systems whose behavior Extemper describes.

Orderings can be achieved in three general ways: exploiting a given ordering, imposing an ordering, or deriving an ordering. In exploiting a given ordering, the connections already present in the knowledge structure are followed to find the next piece of information to say. An ordering must be imposed if the knowledge structure contains no ordering information or if a different ordering is desired. An imposed ordering is usually something that is known to be an effective means of ordering (e.g., describing visual scenes by salience). Deriving an ordering must be done in cases where the underlying knowledge does not contain ordering information (and such information would be expensive to compute) and no imposed ordering exists that can achieve the goals of the elaboration.

The strategy Extemper uses is to exploit the common sense ordering inherent in a knowledge structure. The principle that knowledge should be elaborated in a manner similar to the way it is used is based on the observation that complex transformations of knowledge are not found in naturally occuring elaborations, presumably because they are too time consuming to produce. However, even though Extemper uses a common sense ordering as the primary ordering method, it can be overcome by other factors such as pragmatic goals and listener input (although this has not been done for any of the implemented examples). Also, the design of Extemper's linearizer does not preclude the use of imposed structures for elaboration. In such cases, the linearizer follows an external structure instead of the one internal to the knowledge structure used by the reasoner. This flexibility is possible because Extemper's linearizer is implemented as a set of agenda-based *tasks*. Because these tasks are preemptable, so is the operation of the linearizer, in keeping with Extemper's eventual use in a discourse processing system.

Exploiting a given ordering does not guarantee a complete ordering. A planner, for example, may arbitrarily choose which parts of a plan should be pursued first if the parts independently achieve a goal. The linearizer must, in these cases, decide what parts to express first. This is done using rules that choose among several possible next things to say. For example, in the route planning domain, a goal may be shown to be true simply by reasoning about it, for example, "I am on Olden Street because I am in E-Quad and E-Quad is on Olden Street." and "I assumed you were at Green Hall because you wanted me to meet you there." These two reasoning chains may together contribute to the satisfaction of a goal. Neither one takes precedence over another, so the linearizer uses a rule which prefers the shorter of the two (the second one). Another rule, given several subgoals for a goal, will choose the ones that are supported by reasoning chains over ones that are supported by further planning. Supplementing the ordering produced by the problem solver with (common sense) rules like these gives a complete ordering of concepts for the selectors.

Selection

Relevance is an important constraint on discourse production. Constraints on relevance may be imposed by the overall type of discourse, pragmatic goals, and the characteristics of the domain knowledge. Determining if a given piece of knowledge is relevant is considerably more difficult than selecting a correct ordering for a knowledge structure. Techniques that have been explored include annotations on perceptual-level representations of scenes to indicate visual salience (Conklin, et.al., 1983), and databases containing multiple indexes keyed on user point of view (McKeown, et.al., 1985). However, these techniques involve supplementing *static* knowledge bases and do not readily apply to selecting pertinent information from traces of a problem solver (which Extemper needs for both rehearsal and rehashing). Instead, Extemper uses selectors that rely on different relevance factors to assist in determining relevance "on the fly."

The selectors that Extemper uses are rules constrained by the overall discourse goal of the listener. This enables Extemper to produce elaborations that are tailored to a particular goal (this is the only global relevance factor used in the current implementation). The operation of the selectors is based on the ERKS method for representing meaning, namely that meaning representations consist of a *kernel* (main or core concept) and *nuances* (ancillary concepts that distinguish similar meaning structures). For example, "I drove to Green Hall." is distinguished from "I traveled to Green Hall." by the nuance that, in the second case, the vehicle of transportation was a car. Decomposing meaning in this way allows meaning structures to be built piecewise by the selectors.

Selectors can only perform actions that influence the conceptual content of ERKS meaning structures. They can be divided into three categories: 1) those that select information for inclusion in a meaning structure, 2) those that modify the selected information, and 3) those that add ancillary information to meaning structures. For example, a selector for the route planner might choose to include a concept representing a goal restatement, e.g., "I determined that I could meet you at Green Hall if you and I were there." Another selector (depending upon the overall discourse goal the listener is presumed to have) might choose to modify the concept to omit mention of how this conclusion was arrived at (i.e., omit "I determined that".) A selector that adds ancillary information might produce introductory sentences like the following from the academic counseling domain: "There are four major requirements you must meet for Liberal Arts and Sciences."

The selectors add concepts to one meaning structure until the next concept to be added no longer fits with the ones already selected. In this way, complex sentences are built from separate concepts. The separate concepts are connected according to the relationships found in the knowledge structure (e.g., consecutive actions are joined with a temporal connective). Prior to sentence generation, however, more information must be added to the ERKS meaning structure.

Local Coherence Operators

The local coherence operators annotate ERKS meaning structures to create ties between successive parts of the elaboration. Based on examination of the conceptual forms in the current and previous meaning structures, they make decisions that may influence the syntactic form of the sentence. The most common one is the computation of given and new information, which affects focus. For example, in the route planning domain, if a sentence mentions walking to a place, that (given) location may be the focus for the next sentence.

```
I walked to Green Hall.
At Green Hall, I checked my mail.
```

Another common operation is adding connective words, e.g., using "Ok" to signal the end of a description of how a goal may be achieved in the academic counseling domain.

Local coherence operators are also used to provide certain default values for the meaning structure by filling *annotation nuances*. The most useful of these nuances are shorthand forms for full concepts that express specifications that are extrinsic to the meaning of the kernel concept: absolute time, relative time, and modals. These nuances should be filled by the problem solver, but are not because the problem solvers used do not reason about time or modal information. The English sentence generator needs this information, so it has to be added in this ad hoc manner.

The result of the first three parts, linearization, selection, and local coherence operations, is a set of generation instructions represented as a conceptual form (consisting of a kernel and major nuance concepts) plus annotations on it. This ERKS meaning representation is entered into a *textbase*, a representation of the contents of the discourse. After being added to the textbase, the sentence generator produces English text from it in the manner described in Section 1. Further details on the sentence generator can be found in Cullingford (1986).

The Problem Solving Domain

The emphasis in Extemper's design has been on modeling elaborations of the knowledge and behavior of goal-directed problem solving systems. These problem solvers are typified by systems such as route planners or spatial reasoners. They are of interest because they solve commonplace problems, and their solutions can be described by commonplace (hence extemporaneous) elaborations.

Two basic categories of elaborations are needed for problem solvers, and Extemper produces both kinds. One provides a *rehashing* of something the problem solver has done from a trace of its execution. Rehashed elaborations say what was done to solve the problem, why it was done, and what the results were. The other type of elaboration uses the problem solver's knowledge base directly to elaborate on *rehearsed* knowledge. The elaborator controls a rehearsal (instead of following an execution trace) by alternately rehearsing knowledge and elaborating on it. (Here, rehearsing means fleshing out knowledge structures, e.g., by filling in the value of variables.) Rehearsal is useful for providing information without the overhead of running the problem solver.

Extemper serves as a first step for investigations into methods for producing a variety of elaborations. The problem solving elaborations it produces are descriptions of processes or actions based on the domains of route planning (rehashed elaborations) and academic counseling (rehearsed elaborations). These two domains use very different types of knowledge, from rigid and script-like (the curriculum knowledge in the academic counseling domain) to a mixture of goals, rules and plans (the route planning knowledge). Extemper's ability to interpret these different types of domain knowledge demonstrates its flexibility.

Table 1 shows a sample elaboration produced by Extemper. This description of the route planner's behavior is intended for an operator or programmer of the route planner, and is highly detailed. It gives the justifications underlying the route planner's actions and the "mental states" (e.g., *determined*, *assumed*) of the planner. Extemper also produces two less verbose elaborations from the same knowledge structure, based on different views of the discourse goal of the listener. One of them is shown in Table 2 (the other is simply the last sentence in the elaboration shown in Table 2). In the first, the listener is assumed to be unfamiliar with the area being described, in the second, a high degree of familiarity is assumed. (The output lacks fluency because the focus of this work was not on sentence generation. Some improvement in the output could be gained from more work on sentence generation and pronominalization.)

```
I determined that I could meet you at GreenHall
if you and I were there.

I assumed you were at GreenHall because you
wanted me to meet you there.

I knew I could be at.GreenHall if I was on
the street that it was on, faced it, and
walked to it.

GreenHall was on Washington so I wanted to
be on Washington.

I knew I could be on Washington if I was on
the street that intersects with Washington
and was near the street that I was on,
faced Washington, and walked to the
intersection of Washington and the street
that intersects with Washington and was
near the street that I was on.

William was the street that intersects
with Washington so I wanted to be on
William and to walk to the intersection of
Washington and William.

I assumed I was on William because I was at
EQuad and EQuad was on William.

I assumed I faced Washington because Washington
was oriented west of EQuad, I was at EQuad,
and I faced west.

I walked to the intersection of Washington and
William.

I knew I could face GreenHall if I turned in
the direction that i turn in to be oriented
towards GreenHall.

I turned right and walked to GreenHall.
```

Table 1: A Verbose Elaboration.

```
I could meet you at Green Hall if you and I
were there.

Washington was the street that GreenHall was on
so I wanted to be on Washington.

William was the street that intersects with
Washington so I wanted to be on William and to
walk to the intersection of Washington and
William.

I walked to the intersection of Washington and
William, turned right, and walked to GreenHall.
```

Table 2: A Less Verbose Elaboration.

Related Work

The focus of much current research on language production is on text and discourse planning. Even though Extemper models an extemporaneous process, it must generate lengthy text like text production systems, and be as flexible as discourse planning models in interacting with a user.

HELPCON (a precursor of Extemper) used the conceptual generator CGEN to generate instructions for a computer-aided design system (Cullingford, 1982; Bienkowski, 1983). HELPCON extended CGEN's notion of sketchification at the concept level to the knowledge structure (KS) level by using KS sketchifiers corresponding to different links in *"feature scripts"* that described the CAD tool. HELPCON would traverse a feature script, apply the KS sketchifiers to it, and send the remaining concepts to CGEN. Extemper's use of exploited structure is similar to this.

McKeown's TEXT system (1985) uses "discourse strategies" to link communicative purposes (e.g., define, compare) with the appropriate set of rhetorical techniques (e.g., identify, contrast) for realizing them. The strategies are represented as recursive schemata which, along with the immediate focus of the discourse, impose a partial ordering on the techniques in a given strategy. Schemata are hard to find for some knowledge structures so a more knowledge-driven approach, such as Extemper uses, is needed in some cases.

Mann and Moore's (1981) system, KDS, used a fragment (break input representation into proposition-sized pieces) and compose (combine the resulting propositions) method for generating text from a semantic net representation. The aggregator that KDS uses for combining wastes effort trying useless combinations, a method which could cause problems for large texts. In similar work, Vaughan (1986) has proposed to use a plan and critique cycle to produce text. For modeling extemporaneous language production, however, critiquing is too time-consuming, and more reliable planning mechanisms are needed.

Discourse planning systems treat language as the result of execution of speech acts that are designed to affect another's beliefs or knowledge. These speech acts are planned by modeling their effects on a model of the other's beliefs and knowledge. Appelt (1982), for example, views language production as one of several modalities for a planner of actions. His planner, KAMP, explicitly manipulates assertions involving intensional concepts such as believe and know. Illocutionary acts such as "inform" are axiomatized to capture the intentionality behind them; this enables KAMP to reason about how it can realize its intentions.

Extemper, like discourse planners, integrates the general knowledge an intelligent system has with its language behavior. Discourse planning methods, however, are not applicable to this work because planning an entire elaboration by reasoning about how to affect another's knowledge would overwhelm any planner. Also, reasoning about speech acts does not solve basic problems in elaboration production, since the main relevant speech act for elaboration is only "inform." A planner, rather, would have to reason about how to achieve ordering, relevance and coherence goals in an elaboration. Previous planning systems, then, have operated on a different planning level than is needed for elaborators like Extemper.

The alternative to "planning from scratch" methods is to use schemata or scripts for lengthy discourse production. While these certainly are part of an expert speaker's "bag of tricks," there are several problems with relying on them exclusively for a process like elaboration. One problem is that some elaborations, such as those involving close interaction with a problem solver that is dynamically computing the knowledge structure to be elaborated, cannot be described by schemata. Another is that schema may not be needed for some tasks, if the knowledge structure is sufficiently well-structured.

Extemper does not offer a complete solution to either the problem of making elaboration as flexible as discourse or the problem of finding efficient ways to generate long texts. However, by making use of the knowledge that a problem solving system has, Extemper can guide an elaboration, and its reliance on a flexible task execution methodology (described in Bienkowski, 1986) to do this leaves open the possibility that a method for reasoning about discourse goals can be added in future revisions.

Summary and Conclusions

Extemper's reliance on a reasoning system to guide its processing has been emphasized. In promoting a tight integration between language processing and reasoning components, this work is similar to the work on integrating parsing and memory described by Schank, 1980. Interfacing to a reasoning system requires a translation capability, however, so Extemper does not (in general) generate directly from the structures that the reasoner uses. But it cooperates closely enough so that, if the actions of the reasoner are regarded as the "thoughts" of the machine, Extemper serves as a window on those thoughts.

Extemper relies entirely on conceptual forms throughout its processing cycle. The exclusive use of conceptual forms has been pushed as far as possible to see if the "semantics" (or pragmatics) of a sentence could justify its "syntax." Sometimes this works; other times, nothing in a conceptual form suggests why a particular syntactic form should be needed. This results in non-fluent English. This is not to say that there is no reason for using the forms that could increase fluency, but that the reasons are to be found elsewhere. The work of McKeown (1985) on how tracking focus can affect syntactic structures, is an example of this.

Extemper represents a minimal architecture for extemporaneous elaboration as purposive discourse. The four components it uses are necessary for any elaboration system. They are also sufficient for a certain class of knowledge sources. This work contributes to research on language production by both identifying extemporaneous elaboration as a naturalistic ability that is worthy of study, and by proposing a computational model for it.

REFERENCES

D. E. Appelt. 1985. "Planning English Referring Expressions." **Artificial Intelligence 26**: 1-34.

M. A. Bienkowski. 1983. "Generating Natural Language Explanations." The University of Connecticut, Computer Science TR-CS83-1. Storrs, Connecticut.

M. A. Bienkowski. 1986. "Extemporaneous Elaborations: A Computational Model." Princeton University Cognitive Science Laboratory CSL Report 1. Princeton, New Jersey.

P. R. Cohen and R. C. Perrault. 1979. Elements of a Plan- Based Theory of Speech Acts." **Cognitive Science 3**: 177-212.

E. J. Conklin, et. al. 1983. "An Empirical Investigation of Visual Salience and its Role in Text Generation." **Cognition and Brain Theory**, 6.

R. E. Cullingford, et. al. 1982. "Automated Explanations as a Component of a CAD System." **IEEE Transactions on Systems, Man and Cybernetics**, SMC-12.

R. E. Cullingford. 1986. **Natural Language Processing: A Knowledge Engineering Approach.** Rowman & Littlefield, Inc., Totowa, New Jersey.

Derr, M. A. and McKeown, K. R. 1984. "Using Focus to Generate Complex and Simple Sentences." **Proceedings of the 22nd Annual Meeting of the ACL**. Stanford, California: 319-326.

W. C. Mann and J. A. Moore. 1981. "Computer Generation of Multiparagraph English Text." **American Journal of Computational Linguistics**, 7.

K. R. McKeown, et. al. 1985. "Tailoring Explanations for the User." **Proceedings of the Ninth International Joint Conference on Artificial Intelligence**: Los Angeles, California: 794-798.

K. R. McKeown. 1985. **Text Generation**. Cambridge University Press, Cambridge, United Kingdom.

R. C. Schank, M. Lebowitz, and L. Birnbaum. 1980. "An Integrated Understander," **American Journal of Computational Linguistics**, 6.

T. A. vanDijk and W. Kintsch. 1983. **Strategies of Discourse Comprehension**. Academic Press, New York, New York.

M. M. Vaughan and D. D. McDonald. 1986. "A Model of Revision in Natural Language Generation," **Proceedings of the 24th Meeting of the Association for Computational Linguistics**: New York, New York: 90-95.