Developing and Empirically Evaluating Robust Explanation Generators: The KNIGHT Experiments

James C. Lester[*]
North Carolina State University

Bruce W. Porter[†]
University of Texas at Austin

To explain complex phenomena, an explanation system must be able to select information from a formal representation of domain knowledge, organize the selected information into multi-sentential discourse plans, and realize the discourse plans in text. Although recent years have witnessed significant progress in the development of sophisticated computational mechanisms for explanation, empirical results have been limited. This paper reports on a seven year effort to empirically study explanation generation from semantically rich, large-scale knowledge bases. In particular, it describes KNIGHT*, a robust explanation system that constructs multi-sentential and multi-paragraph explanations from the* Biology Knowledge Base*, a large-scale knowledge base in the domain of botanical anatomy, physiology, and development. We introduce the* Two Panel *evaluation methodology and describe how* KNIGHT*'s performance was assessed with this methodology in the most extensive empirical evaluation conducted on an explanation system. In this evaluation,* KNIGHT *scored within "half a grade" of domain experts, and its performance exceeded that of one of the domain experts.*

1. Introduction

In the course of their daily affairs, scientists explain complex phenomena—both to one another and to lay people—in a manner that facilitates clear communication. Similarly, physicians, lawyers, and teachers are equally facile at generating explanations in their respective areas of expertise. In an effort to computationalize this critical ability, research in natural language generation has addressed a broad range of issues in automatically constructing text from formal representations of domain knowledge. Research on text planning (Hovy, 1993; Maybury, 1992; McCoy, 1989 1990; McKeown, 1985; Paris, 1988) has developed techniques for determining the content and organization of many genres, and explanation generation (Cawsey, 1992; McKeown, Wish, and Matthews, 1985; Moore, 1995) in particular has been the subject of intense investigation. In addition to exploring a panorama of application domains, the explanation community has begun to assemble these myriad designs into a coherent framework. As a result, we have begun to see a crystalization of the major components, as well as detailed analyses of their roles in

[*] Department of Computer Science, Box 8206, North Carolina State University, Raleigh, NC 27695-8206. E-mail: lester@adm.csc.ncsu.edu
[†] Department of Computer Sciences, University of Texas at Austin, Austin, Texas 78712-1188. E-mail: porter@cs.utexas.edu

explanation (Suthers, 1991).

Despite this success, empirical results in explanation generation are limited. Although techniques for developing and evaluating robust explanation generation should yield results that are more conclusive than those produced by prototype, "proof-of-concept" systems, with only a few notable exceptions (Cawsey, 1992; Hovy, 1990; Kukich, 1983; Mittal, 1993; Robin, 1994), most work has adopted a research methodology in which a proof-of-concept system is constructed and its operation is analyzed on a few examples. While isolating one or a small number of problems enables researchers to consider particular issues in detail, it is difficult to gauge the scalability and robustness of a proposed approach.

A critical factor contributing to the dearth of empirical results is the absence of semantically rich, large-scale knowledge bases. Knowledge bases housing tens of thousands of different concepts and hundreds of different relations could furnish ample raw materials for empirical study, but no work in explanation generation has been conducted or empirically evaluated in the context of these knowledge bases.

To empirically study explanation generation from semantically rich, large-scale knowledge bases, we undertook a seven year experiment. First, our domain experts (one employed full-time) constructed the Biology Knowledge Base (Porter et al., 1988), a very large structure representing more than 180,000 facts about botanical anatomy, physiology, and development. Second, we designed, implemented, and empirically evaluated KNIGHT (Lester, 1994), a robust explanation system that extracts information from the Biology Knowledge Base, organizes it, and realizes it in multi-sentential and multi-paragraph expository explanations of complex biological phenomena. Third, we developed a novel evaluation methodology for gauging the effectiveness of explanation systems and employed this methodology to evaluate KNIGHT.

This paper describes the lessons learned during the course of the "KNIGHT experiments." In the spirit of EDGE (Cawsey, 1992) and PAULINE (Hovy, 1990), which synthesize work in interactive explanation systems and generational pragmatics, respectively, KNIGHT addresses a broad range of issues, all in the context of semantically rich, large-scale knowledge bases:

- *Robust Knowledge-Base Access:* KNIGHT exploits a library of robust knowledge-base access methods that insulate discourse planners from the idiosyncracies and errors in knowledge bases. These "view construction" methods selectively extract coherent packets of propositions about the structure and function of objects, the changes made to objects by processes, and the temporal attributes and temporal decompositions of processes.
- *Discourse-Knowledge Engineering:* Discourse-knowledge engineers, i.e., knowledge engineers who encode discourse knowledge, should be able to inspect and easily modify discourse-planning specifications for rapid iterative refinement. The *Explanation Design Package* (EDP) formalism is a convenient, schema-like (McKeown, 1985; Paris, 1988) programming language for text planning. Because the EDP formalism is a hybrid of the declarative and procedural paradigms, discourse-knowledge engineers can easily understand EDPs, modify them, and use them to represent new discourse knowledge. EDPs have been used by KNIGHT to generate hundreds of expository explanations of biological objects and processes.
- *Explanation Planning:* KNIGHT employs a robust explanation planner that selects EDPs and applies them to invoke knowledge-base accessors. The explanation planner considers the desired length of explanations and the

relative importance of sub-topics as it constructs explanation plans encoding content and organization.

- *Functional Realization:* KNIGHT's functional realization system (Callaway and Lester, 1995) is built on top of a unification-based surface generator with a large systemic grammar (Elhadad, 1992).

To assess KNIGHT's performance, we developed the *Two-Panel Evaluation Methodology* for natural language generation and employed it in the most extensive and rigorous empirical evaluation ever conducted on an explanation system. In this study, KNIGHT constructed explanations on randomly chosen topics from the Biology Knowledge Base. A panel of domain experts was instructed to produce explanations on these same topics, and both KNIGHT's explanations and the explanations produced by this panel were submitted to a second panel of domain experts. The second panel then graded all of the explanations on several dimensions with an A–F scale. KNIGHT scored within approximately "half a grade" of the domain experts, and its performance exceeded that of one of the domain experts.

This paper is structured as follows. The task of explanation generation is characterized and the Biology Knowledge Base is described. A brief description of KNIGHT's knowledge-base access methods is followed by (1) a description of the EDP language, (2) KNIGHT's explanation planner, and (3) an overview of the realization techniques. The empirical evaluation is then discussed in some detail. The paper concludes with discussions of related work and future research directions.

2. The Task of Explanation Generation

Explanation generation is the task of extracting information from a formal representation of knowledge, imposing an organization on it, and realizing the information in text. An explanation system must be able to map from a formal representation of domain knowledge, i.e., one which can be used for automated reasoning, such as the predicate calculus, to a textual representation of domain knowledge. Because of the significant differences in formal and textual representational schemes, successfully bridging the gap between them is one of the major challenges faced by an explanation system.

To communicate complex ideas, an explanation system should be able to produce extended explanations such as those in Figure 1, which shows several explanations produced by KNIGHT from the domain of botanical anatomy, physiology, and development. Note that each of these is a *multi-sentential* explanation; the first is a *multi-paragraph* explanation. These explanations are expository: in contrast to causal explanations produced by automated reasoning systems, expository explanations describe domain phenomena, such as anatomical structures and physiological processes.

Explanation generation is typically decomposed into two subtasks, *explanation planning* and *realization*. Explanation planning itself has two subtasks: *content determination*, in which knowledge structures are extracted from a knowledge base, and *organization*, in which the selected knowledge structures are arranged in a manner appropriate for communication in natural language. To perform these tasks successfully, an explanation planner must have access to *discourse knowledge*, which informs its decisions about the content and organization of textual explanations. The organizational aspect of discourse knowledge plays a particularly important role in the construction of extended explanations. Our work focuses on content determination and organization and de-emphasizes issues in realization. However, to more cleanly evaluate the explanation planning work, we have developed a robust realization system that is built on FUF (Elhadad, 1992), a

Question: What happens during *embryo sac formation*?

Response: Embryo sac formation is a kind of female gametophyte formation. During embryo sac formation, the embryo sac is formed from the megaspore mother cell. Embryo sac formation occurs in the ovule.

Embryo sac formation is a step of angiosperm sexual reproduction. It consists of megasporogenesis and embryo sac generation. During megasporogenesis, the megaspore mother cell divides in the nucellus to form 4 megaspores. During embryo sac generation, the embryo sac is generated from the megaspore.

Question: What happens during *pollen tube growth*?

Response: Pollen tube growth is a step of angiosperm sexual reproduction. It consists of sperm cell generation and sperm cell transport. During sperm cell generation, the pollen generative cell divides to form 2 angiosperm sperm cells. During sperm cell transport, 2 angiosperm sperm cells are transported from the pollen tube to the embryo sac.

Question: What is a *spore*?

Response: The spore is a kind of haploid cell. 4 spores are produced from the spore mother cell during sporogenesis. The spore divides to form 2 plant gametes during gametogenesis. Gametogenesis is a step of gametophyte development.

Question: What is a *root system*?

Response: The root system is part of the plant and is connected to the mainstem. It is below the hypocotyl and is surrounded by the rhizosphere. The subregions of the root system include the meristem, which is where root system growth occurs.

Figure 1
Explanations produced by KNIGHT from the Biology Knowledge Base

unification-based implementation of a large systemic grammar.

2.1 Evaluation Criteria and Desiderata

Evaluating the performance of explanation systems is a critical and non-trivial problem. Although gauging the performance of explanation systems is inherently difficult, five evaluation criteria can be applied.

- *Coherence*: a global assessment of the overall quality of the explanations generated by a system,

- *Content*: the extent to which explanations' information is adequate and focused,
- *Organization*: the extent to which the information is well organized,
- *Writing style*: the quality of the prose, and
- *Correctness*: for scientific explanations, the extent to which the explanations are in accord with the established scientific record.

In addition to performing well on the evaluation criteria, if explanation systems are to make the difficult transition from research laboratories to fielded applications, we want them to exhibit two important properties, both of which significantly affect scalability. First, these systems' representation of discourse knowledge should be easily inspected and modified. To develop explanation systems for a broad range of domains, tasks, and question types, discourse-knowledge engineers must be able to create and efficiently debug the discourse knowledge that drives the systems' behavior. The second property that explanation systems should exhibit is robustness. Despite the complex and possibly mal-formed representational structures that an explanation system may encounter in its knowledge base, it should be able to cope with these structures and construct reasonably well-formed explanations.

2.2 Semantically Rich, Large-Scale Knowledge Bases

Given the state of the art in explanation generation, the field is now well positioned to explore what may pose its greatest challenge and at the same time may result in its highest payoff: generating explanations from semantically rich, large-scale knowledge bases. Large-scale knowledge bases encode information about domains that cannot be reduced to a small set of principles or axioms. For example, the field of human anatomy and physiology encompasses a body of knowledge so immense that many years of study are required to assimilate only one of its subfields, such as immunology. Large-scale knowledge bases are currently being constructed for many applications, and the ability to generate explanations from these knowledge bases for a broad range of tasks such as education, design and diagnosis is critical.

Large-scale knowledge bases whose representations are semantically rich are particularly intriguing. These knowledge bases consist of highly interconnected networks of (at least) tens of thousands of facts. Hence, they represent information not only about a large number of concepts but also about a large number of relationships that hold between the concepts. One such knowledge base is the **Biology Knowledge Base** (Porter et al., 1988), an immense structure encoding information about botanical anatomy, physiology, and development. One of the largest knowledge bases in existence, it is encoded in the KM frame-based knowledge representation language.[1] KM provides the basic functionalities of other frame-based representation languages and is accompanied by a graphical user interface, KnEd, for entering, viewing, and editing frame-based structures (Eilerts, 1994).

The backbone of the Biology Knowledge Base is its taxonomy, which is a large hierarchical structure of biological objects and biological processes. In addition to the objects and processes, the taxonomy also includes the hierarchy of relations that may appear on concepts. The relation taxonomy provides a useful organizing structure for encoding information about "second order" relations, i.e., relations among all of the first

1 A detailed description of the semantics of the representation language may be found in Ch. 2 of (Acker, 1992).

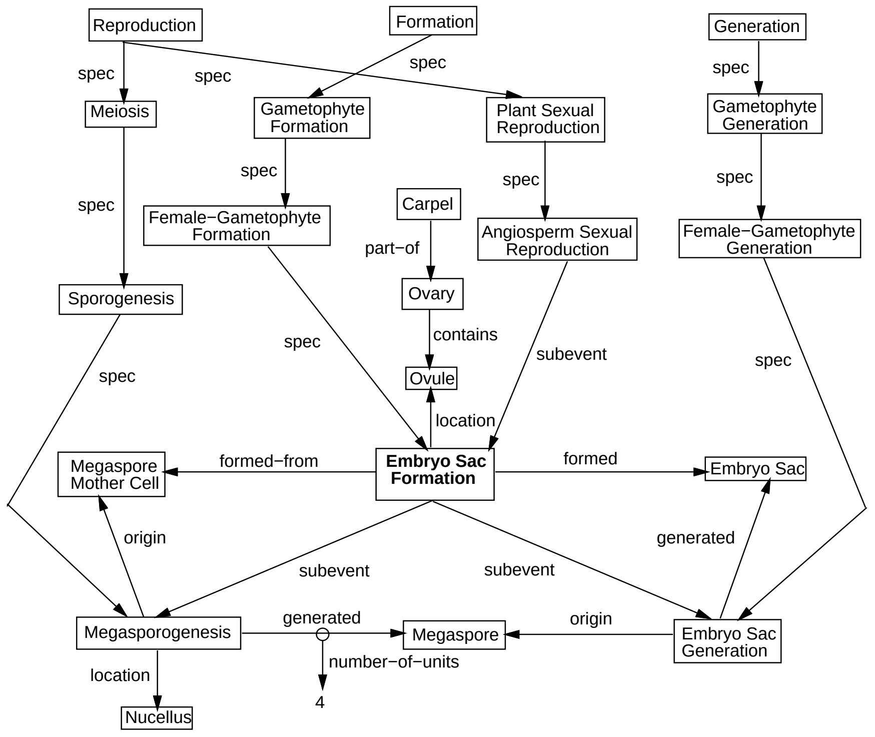


Figure 2
A representation of embryo sac formation

order relations.

Figure 2 depicts the Biology Knowledge Base's representation of *embryo sac formation*. This is a typical fragment of its semantic network. Each of the nodes in this network is a concept, e.g. *megaspore mother cell*, which we refer to as a "unit" or a "frame." Each of the arcs is a relation in the knowledge base. For example, the *location* for *embryo sac formation* is the concept *ovule*. We refer to these relations as "slots" or "attributes" and to the units that fill these slots, e.g., *ovule*, as "values." In addition, we term a structure of the form (*Unit Slot Value*) as a "triple." The Biology Knowledge Base currently contains more than 180,000 explicitly represented triples, and its deductive closure is significantly larger.

We chose biology as a domain for three reasons. First, it required us to grapple with difficult representational problems. Unlike a domain such as introductory geometry, biology cannot be characterized by a small set of axioms. Second, biology is not a "single-task" subject. Unlike the knowledge bases of conventional expert systems, e.g., MYCIN (Buchanan and Shortliffe, 1984), the Biology Knowledge Base is not committed to any particular task or problem-solving method. Rather, it encodes general knowledge that can support diverse tasks and methods such as tutoring students, performing diagnosis, and organizing reference materials. For example, in addition to its use in explanation generation, it has been used as the basis for an automated qualitative model builder (Rickel and Porter, 1994) for qualitative reasoning. Finally, we chose biology because of the availability of local domain experts at the University of Texas at Austin.

It is important to note that the authors and the domain experts entered into a "contractual agreement" with regard to representational structures in the Biology Knowledge Base. To eliminate all requests for representational modifications that would skew the knowledge base to the task of explanation generation, the authors entered into this agreement: they could request representational changes only if knowledge was inconsistent or missing. This facilitated a unique experiment in which the representational structures were not tailored for the task of explanation generation.

3. Accessing Semantically Rich, Large-Scale Knowledge Bases

To perform well, an explanation system must select from a knowledge base precisely that information needed to answer users' questions with coherent and complete explanations. Given the centrality of content determination for explanation generation, it is instructive to distinguish two types of content determination, both of which play key roles in an explanation system's behavior. "Local" content determination is the selection of relatively small knowledge structures, each of which will be used to generate one or two sentences; "global" content determination is the process of deciding which of these structures to include in an explanation.

There are two benefits of interposing a KB accessing system between an explanation planner, which performs global content determination, and a knowledge base. First, it keeps the explanation planner at "arm's length" from the representation of domain knowledge, thereby making the planner less dependent on the particular representational conventions of the knowledge base and more robust in the face of errors. In addition, it can help build explanations that are coherent. Studies of coherence have focused on one aspect of coherence, *cohesion*, which is determined by the overall organization and realization of the explanation (Grimes, 1975; Halliday and Hassan, 1976; Hobbs, 1985; Joshi and Weinstein, 1981). However, an equally important question is, "To insure coherence, how should the content of individual portions of an explanation be selected?" Halliday and Hassan (Halliday and Hassan, 1976) term this aspect of coherence *semantic unity*. There are at least two approaches to achieving semantic unity: either "packets" of propositions must be directly represented in the domain knowledge, or a KB accessing system must be able to extract them at runtime.

One type of coherent knowledge packet is a *view*. For example, the concept *photosynthesis* can be viewed as either a *production* process or an *energy transduction process*. Viewed as *production*, it would be described in terms of its *raw materials* and *products*: "During photosynthesis, a chloroplast uses water and carbon dioxide to make oxygen and glucose." Viewed as *energy transduction*, it would be described in terms of *input energy forms* and *output energy forms*: "During photosynthesis, a chloroplast converts light energy to chemical bond energy." The view that is taken of a concept has a significant effect on the content that is selected for its description. If an explanation system could (a) invoke a knowledge base accessing system to select views, and (b) translate the views to natural language (Figure 3), it would be well on its way to producing coherent explanations.

As a building block for the KNIGHT explanation system, we designed and implemented a robust KB accessing system that extracts *views* (Acker, 1992; McCoy, 1989 1990; McKeown, Wish, and Matthews, 1985; Souther et al., 1989; Swartout, 1983; Suthers, 1988; Suthers, 1993) of concepts represented in a knowledge base. Each view is a coherent subgraph of the knowledge base describing the structure and function of objects, the change made to objects by processes, and the temporal attributes and temporal decompositions of processes. Each of the nine accessors in our library (Table 1) can be applied to a given concept (the "concept of interest") to retrieve a view of that concept. There are three classes of Accessors: those that are applicable to all concepts (*As-Kind-Of* and *Functional*), those that are applicable to objects (*Partonomic-Connection* and *Sub-Structural*), and those that are applicable to processes (*Auxiliary-Process*—which

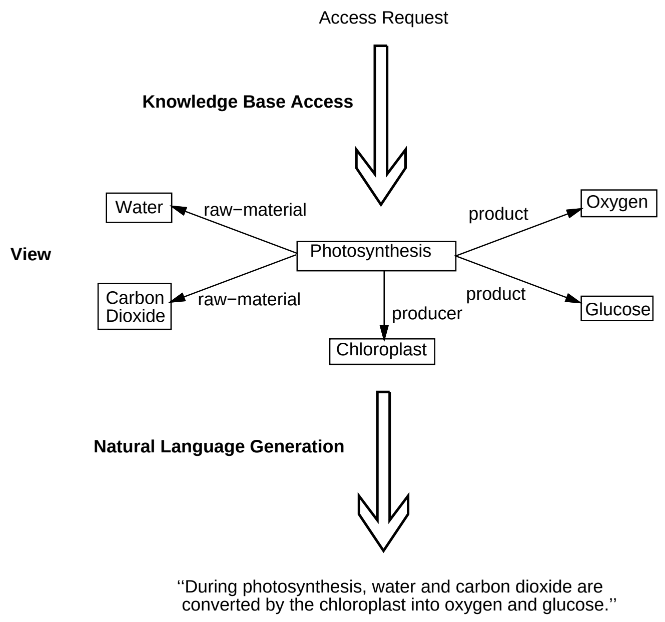


Figure 3
Accessing and translating a view of photosynthesis

includes *Causal*, *Modulatory*, *Temporal*, and *Locational* sub-types—*Participants*, *Core-Connection*,[2] and *Sub-event*, and *Temporal-Step*). In addition to the "top level" accessors, the library also provides a collection of some twenty "utility" accessors that extract particular aspects of views previously constructed by the system.[3]

To illustrate, the *Participants* accessor extracts information about the "actors" of the given process. For example, some of the actors in the Photosynthesis process are Chloroplasts, Light, Chlorophyll, Carbon Dioxide, and Glucose. By specifying a *reference process*—the second argument to the *Participants* accessor—the external agent can request a view of the process from the perspective of the reference process. For example, if the system applies the *Participants* accessor with Photosynthesis as the concept of interest and Production as the reference process, then the accessor will extract information about the producer (Chloroplast), the raw materials (Water and Carbon Dioxide), and the products (Oxygen and Glucose). In contrast, if the system applies the *Participants* accessor with Photosynthesis as the concept of interest but with Energy Transduction as the reference process, then it would extract information about the transducer (Chlorophyll), the energy provider (a Photon), the input energy form (Light), and the output energy form (Chemical Bond Energy). By selecting different reference concepts, different information about a particular process will be returned.

In addition to coherence, robustness is also an important design criterion. We define *robustness* as the ability to gracefully cope with the complex representational structures encountered in large-scale knowledge bases without failing (halting execution). The KB

2 The *Core Connection* Accessor determines the relation between a process and a *core* concept. A core concept is one which is particularly central to a domain. For example, in biology, processes such as "development" and "reproduction" play central roles in many physiological explanations. During the design of a KB accessing system, the domain-knowledge engineer selects the core concepts and flags these concepts in the knowledge base.

3 For a more comprehensive description of the accessors, see (Lester, 1994).

Table 1
Library of Knowledge Base Accessors

KB Accessor	*Arguments*	*Description of View*
As-Kind-Of	*concept* *reference*	Finds view of *concept* as a kind of *reference* concept.
Auxiliary-Process	*process* *view-type*	Finds temporal, causal, or locational information about *process* as specified by *view-type*.
Participants	*process* *reference*	Finds "actor-oriented" view of *process* as viewed from the perspective of *reference* process.
Core-Connection	*process*	Finds the connection between *process* and a "core" process.
Functional	*object* *process*	Finds functional view of *object* with respect to *process*.
Partonomic-Connection	*object*	Finds the connection from *object* to a "superpart" of the object in the "partonomy."
Subevent	*process*	Finds view of "steps" of *process*.
Sub-Structural	*object*	Finds structural view of parts of *object*.
Temporal-Step	*process*	Finds view of *process* with respect to another process of which *process* is a "step."

accessors achieve robust performance in four ways:

- *Omission Toleration:* They do not assume that essential information will actually appear on a given concept in the knowledge base.
- *Type Checking:* They employ a type checking system that exploits the knowledge base's taxonomy.
- *Error Handling:* When they detect an irregularity, they return appropriate error codes to the explanation planner.
- *Term Accommodation:* They tolerate specialized (and possibly unanticipated) representational vocabulary by exploiting the relation taxonomy.

The following four techniques operate in tandem to achieve robustness. First, to cope with knowledge structures that contain additional, unexpected information, the KB Accessors were designed to behave as "masks." When they are applied to particular structures in a knowledge base, the Accessors mask out all attributes that they were not designed to seek. Hence, they are unaffected by inappropriate attributes that were installed on a concept erroneously. Second, sometimes a domain-knowledge engineer installs inappropriate values on legal attributes. When the Accessors encounter attributes with inappropriate values, they prevent fatal errors from occurring by employing a rigorous type checking system. For example, suppose a domain-knowledge engineer had erroneously installed an object as one of the subevents of a process. The type checking system detects the problem. Third, when problems are detected, the nature of the error is noted and reported to the explanation planner. Because the planner can reason about the types of problems, it can properly attend to them by excising the offending content from the explanation it is constructing. The KB Accessor library currently uses more than 25 different error codes to report error conditions. For example, it will report *no superevent available* if the "parent" event of a process has not been included. Fourth, the KB Accessors exhibit immunity to modifications of the representational vocabulary by the domain-knowledge engineer. For example, given an object, the *Sub Structural* Accessor inspects the object to determine its parts. Rather than merely examining the attribute *parts* on the given object, the *Sub Structural* Accessor examines all known attributes that bear the *parts* relation to other objects. These attributes include *has basic unit*, *layers*, *fused parts*, and *protective components*. The *Sub Structural* Accessor recognizes that each of these attributes are partonomic relations by exploiting the knowledge base's relation taxonomy.

By using these techniques together, we have developed a KB accessing system that has constructed several thousand views without failing. Moreover, the view types on which the accessors are based performed well in a preliminary empirical study (Acker and Porter, 1994), and evaluations of the KB accessors' ability to construct coherent views, as measured by domain experts' ratings of KNIGHT's explanations (Section 8), are encouraging.

4. A Programming Language for Discourse Knowledge

Since the time of Aristotle, a central tenet of rhetoric has been that a rich structure underlies text. This structure shapes a text's meaning and assists its readers in deciphering that meaning. For almost two decades, computational linguists have studied the problem of automatically inducing this structure from a given text. Research in explanation planning addresses the inverse problem: automatically creating this structure by selecting facts from a knowledge base and subsequently using these facts to produce text. To automatically construct *explanation plans* (trees that encode the hierarchical structure of texts, as well as their content (Grosz and Sidner, 1986; Mann and Thompson, 1987)), an explanation system must possess knowledge about what characterizes a clear explanation. This *discourse knowledge* enables it to make decisions about what information to include in its explanations and how to organize the information.

It is important to emphasize the following distinction between discourse knowledge and explanation plans. While discourse knowledge specifies the content and organization for a *class* of explanations, e.g., explanations of processes, explanation plans specify the content and organization for a *specific* explanation, e.g., an explanation of how photosynthesis produces sugar. Discourse-knowledge engineers build representations of discourse knowledge, and this discourse knowledge is then used by a computational module to automatically construct explanation plans, which are then interpreted by a realization system to produce natural language.

The KB accessing system described above possesses discourse knowledge in the form of KB Accessors. Applying this discourse knowledge, the system retrieves views from the knowledge base. Although this ability to perform local content determination is essential, it is insufficient; given a query posed by a user, the generator must be able to choose multiple KB accessors, provide the appropriate arguments to these accessors, and organize the resulting views. Hence, in addition to discourse knowledge about local content determination, an explanation system that produces multi-paragraph explanations must also possess knowledge about how to perform global content determination and organization. This section sets forth two design requirements for a representation of discourse knowledge, describes the *Explanation Design Package* (EDP) formalism, which was designed to satisfy these requirements, and discusses how EDPs can be used to encode discourse knowledge.

4.1 Requirements for a Discourse Knowledge Representation

Our goal is to develop a representation of discourse knowledge that satisfies two requirements: It should be expressive, and it should facilitate efficient representation of discourse knowledge by discourse-knowledge engineers.[4] Each of these considerations are discussed in turn, followed by a representation that satisfies these criteria.

Expressiveness. A representation of discourse knowledge must permit discourse-knowledge engineers to state how an explanation planner should

- select propositions from a knowledge base by extracting views,
- control the amount of detail in an explanation, i.e., if a user requests that terse explanations be generated, the explanation planner should select only the most important propositions,
- consider contextual conditions when determining which propositions to include,
- order the propositions, and
- group the propositions into appropriate segments, e.g., paragraphs.

The first three aspects of expressiveness are concerned with content determination. To effectively express what content should be included in explanations, a representation of discourse knowledge should enable discourse-knowledge engineers to encode specifications about how to choose propositions about particular topics, the importance of those topics, and under what conditions the propositions associated with the topics should be included. These "inclusion conditions" govern the circumstances under which the explanation planner will select particular classes of propositions from the knowledge base when constructing an explanation.

For example, a discourse-knowledge engineer might express the rule: "The system should communicate the location of a process if and only if the user of the system is familiar with the object where the process occurs." As the explanation planner uses this knowledge to construct a response, it can determine if the antecedent of the rule ("the user of the system is familiar with the object where the process occurs") is satisfied by the current context; if the antecedent is satisfied, then the explanation planner can include in the explanation the subtopics associated with the rule's consequent.

The final two aspects of expressiveness (ordering and grouping of propositions) are concerned with organization. To encode organizational knowledge, a representation of discourse knowledge should permit discourse-knowledge engineers to encode topic/subtopic relationships. For example, the subtopics of a process description might include (1) a categorical description of the process (describing taxonomically what kind of process it is), (2) how the "actors" of the process interact, and (3) the location of the process.

A representation should be sufficiently expressive that it can be used to encode the kinds of discourse knowledge discussed above, and it should be applicable to representing discourse knowledge for a broad range of discourse genres and domains. However, discourse knowledge does not specify what syntactic structure to impose on a sentence; nor does it lend any assistance in making decisions about matters such as pronominalization, ellipsis, or lexical choice. These decisions are delegated to the realization system.

4 While expressiveness and knowledge engineering are the criteria we address, others are also of considerable importance, e.g., soundness and completeness of discourse planners.

Discourse-Knowledge Engineering. For a given query type, domain, and task, a discourse-knowledge engineer must be able to represent the discourse knowledge needed by an explanation system for responding to questions of that type in that domain about that task. Pragmatically, to represent discourse knowledge for a broad range of queries, domains, and tasks, a formalism must facilitate *efficient* representation of discourse knowledge. Kittredge *et al* have observed that representing new domain-dependent discourse knowledge—they term it "domain communication knowledge"—is required to create advanced discourse generators, e.g., those for special purpose report planning (Kittredge, Korelsky, and Rambow, 1991). Therefore, ease of creation, modification, and reuse are important goals for the design of a discourse formalism.

For example, to build an explanation system for the domain of physics, a discourse-knowledge engineer could either build an explanation system *de novo* or modify an existing system. On the face of it, the second alternative involves less work and is preferable, but designing explanation systems that can be easily modified is a non-trivial task. In the case of physics, a discourse-knowledge engineer may need to modify an existing explanation system so that it can produce explanations that are appropriate for mathematical explanations. To do so, the discourse-knowledge engineer would ideally take an off-the-shelf explanation generator and add discourse knowledge about how to explain mathematical interpretations of the behavior of physical systems. Because of the central role played by discourse-knowledge engineers, a representation of discourse knowledge should be designed to minimize the effort required to understand, modify, and represent new discourse knowledge.

4.2 Explanation Design Packages

Explanation Design Packages emerged from an effort to accelerate the representation of discourse knowledge without sacrificing expressiveness. Our previous explanation generators employed a representation of discourse knowledge that was coded directly in Lisp (Lester and Porter, 1991a; Lester and Porter, 1991b). Although this approach worked well for small prototype explanation systems, it proved unsatisfactory for building fully functioning explanation systems. In particular, it was very difficult to maintain and extend discourse knowledge expressed directly in code.

Although EDPs are more schema-like than plan-based approaches and consequently do not permit an explanation system to reason about the goals fulfilled by particular text segments[5], they have proven enormously successful for discourse-knowledge engineering. EDPs give discourse-knowledge engineers an appropriate set of abstractions for specifying the content and organization of explanations. They combine a frame-based representation language with embedded procedural constructs. To mirror the structure of expository texts, an EDP contains a hierarchy of nodes, which provides the "global organization" of explanations. EDPs are schema-like (McKeown, 1985; Paris, 1988) structures that include constructs found in traditional programming languages. Just as prototypical programming languages offer conditionals, iterative control structures, and procedural abstraction, EDPs offer discourse-knowledge engineers counterparts of these constructs that are precisely customized for explanation planning.[6] Moreover, each EDP names multiple KB accessors, which are invoked at explanation planning time.

Because EDPs are frame-based, they can be easily viewed and edited by knowledge engineers using the graphical tools commonly associated with frame-based languages. The EDP formalism has been implemented in the Km frame-based knowledge representation

5 See Section 9 for a discussion of this disadvantage.
6 EDPs are Turing-equivalent.

Table 2
EDP Node Attributes

Node Type	*Attributes*	*Attribute Value(s)*
Exposition	Children	⟨Topics⟩
Topic	Children	⟨Content Specifications⟩
	Centrality	{Low, Medium, High}
	Inclusion Condition	⟨Variable Boolean Expression⟩
	Local Variables	(⟨Var⟩ , ⟨Variable Expr.⟩) Pairs
Content Specification	Children	{⟨Content Spec's⟩, ⟨Elaborations⟩}
	Content Specification Template	⟨Variable Expression with KB Accessor⟩
	Iteration Type	{Non-Iter., Iter., Conditional-Iter.}
	Iterate-Over Template	⟨Variable Expression⟩
	Loop Variable	⟨Var⟩
	Iteration Condition	⟨Variable Boolean Expression⟩
	Local Variables	(⟨Var⟩ , ⟨Variable Expr.⟩) Pairs
Elaboration	Children	⟨Content Specifications⟩
	Centrality	{Low, Medium, High}
	Inclusion Condition	⟨Variable Boolean Expression⟩
	Local Variables	(⟨Var⟩ , ⟨Variable Expr.⟩) Pairs

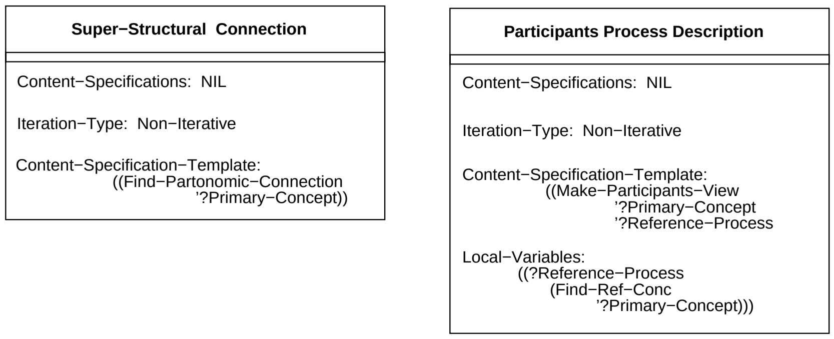
language, which is the same representational language used in the Biology Knowledge Base. Because KM is accompanied by a graphical user interface, discourse-knowledge engineers are provided with a development environment that facilitates EDP construction. This has proven to be very useful for addressing a critical problem in scaling up explanation generation: maintaining a knowledge base of discourse knowledge that can be easily constructed, viewed, and navigated by discourse-knowledge engineers.

EDPs have several types of nodes, where each type provides a particular set of attributes to the discourse-knowledge engineer (Table 2). Note that content specification nodes may have elaboration nodes as their children, which in turn may have their own content specification nodes. This "recursive" appearance of content specification nodes permits a discourse-knowledge engineer to construct arbitrarily deep trees. In general, a node of a particular type in an EDP is used by the explanation planner to construct a corresponding node in an explanation plan. We discuss the salient aspects of each type of node below.[7]

Exposition Nodes. An exposition node is the top-level unit in the hierarchical structure and constitutes the highest-level grouping of content. For example, the exposition node of the *Explain-Process* EDP has four children, *Process Overview*, *Output-Actor-Fates*, *Temporal Info*, and *Process Details*, each of which is a topic node. Both the order and grouping of the topic nodes named in an exposition node are significant. The order specifies the linear left-to-right organization of the topics, and the grouping specifies the paragraph boundaries. The content associated with topic nodes that are grouped together will appear in a single paragraph in an explanation.

Topic Nodes. Topic nodes are "subtopics" of exposition nodes, and each topic node includes a representation of the conditions under which its content should be added to an

7 Representational details of EDPs are discussed in (Lester, 1994).



```
Super−Structural  Connection

Content−Specifications:  NIL

Iteration−Type:  Non−Iterative

Content−Specification−Template:
                ((Find−Partonomic−Connection
                              '?Primary−Concept))
```

```
Participants Process Description

Content−Specifications:  NIL

Iteration−Type:  Non−Iterative

Content−Specification−Template:
            ((Make−Participants−View
                       '?Primary−Concept
                       '?Reference−Process

Local−Variables:
        ((?Reference−Process
             (Find−Ref−Conc
                    '?Primary−Concept)))
```

Figure 4
Example content specification nodes

explanation. Topic nodes have the *atomic inclusion* property which enables an explanation planner to make an "atomic" decision about whether to include—or exclude—all of the content associated with a topic node. Atomicity permits discourse-knowledge engineers to achieve coherence by demanding that the explanation planner either include or exclude all of a topic's content. At runtime, if the explanation planner determines that inclusion conditions are not satisfied or if a topic is not sufficiently important given space limitations (see below), it can comprehensively eliminate all content associated with the topic.

An important aspect of discourse knowledge is the relative importance of subtopics with respect to one another. If an explanation's length must be limited—such as when a user has employed the verbosity preference parameter to request terse explanations—an explanation planner should be able to decide at runtime which propositions to include. EDPs permit discourse-knowledge engineers to specify the relative importance of each topic by assigning a qualitative value (`Low`, `Medium`, or `High`) to its centrality attribute.

Another important aspect of representing discourse knowledge is the ability to encode the conditions under which a group of propositions should be included in an explanation. Discourse-knowledge engineers can express these *inclusion conditions* as predicates on the knowledge base and on a user model (if one is employed). For example, he or she should be able to express the condition that the content associated with the *Output Actor Fates Topic* should be included only if the process being discussed is a conversion process. Inclusion conditions are expressed as boolean expressions that may contain both built-in user modeling predicates and user-defined functions.

Content Specification Nodes. Content specification nodes house the high-level specifications for extracting content from the knowledge base. To fulfill this function, they provide constructs known as *content specification expressions.* These expressions are instantiated at runtime by the explanation planner, which then dispatches the Knowledge Base Accessors named in the expressions to extract propositions from the knowledge base. Content specification expressions reside in content specification nodes, e.g., Figure 4. When creating content specification expressions, the discourse-knowledge engineer may name any Knowledge Base Accessor in the KB Accessor Library. For example, the *Super Structural Connection* content specification in Figure 4 names a KB Accessor called *Find Partonomic Connection*, and the *Process Participants Description* content specification names the *Make-Participants-View* Accessor.

Although the discourse-knowledge engineer may write arbitrarily complex specification expressions in which function invocations are deeply nested, these expressions can become difficult to understand, debug, and maintain. Just as other programming languages provide local variables, e.g., the binding list of a `let` statement in Lisp, so do content specification nodes. Each time a discourse-knowledge engineer creates a local variable, he or she creates an expression for computing the value of the local variable at runtime. For example, the *Process Participants Description* content specification in Figure 4 employs a local variable *?Reference Process*. The content specification expression associated with *?Reference Process* names the KB Accessor *Find Ref Conc* and the global variable *?Primary-Concept*. Local variables provide a means for decomposing more complex content specification expressions into simpler ones.

Elaboration Nodes. Elaboration nodes specify optional content that may be included in explanations. They are structurally and functionally identical to topic nodes, i.e., they have exactly the same attributes, and the children of elaboration nodes are content specifications. The distinction between elaboration nodes and topic nodes is maintained only as a conceptual aid to discourse-knowledge engineers: it stands as a reminder that topic nodes are used to specify the primary content of explanations, and elaboration nodes are used to specify supplementary content.

4.3 Developing Task-Specific EDPs

A discourse-knowledge engineer can use EDPs to encode discourse knowledge for his or her application. In our work, we focused on two types of texts that occur in many domains: *process descriptions* and *object descriptions*. For example, in biology, one encounters many process-oriented descriptions of physiological and reproductive mechanisms, as well as many object-oriented descriptions of anatomy. In the course of our research, we informally reviewed numerous (on the order of one hundred) passages in several biology textbooks. These passages focused on explanations of the anatomy, physiology, and reproduction of plants. Some explanations were very terse, e.g., those that occurred in glossaries, whereas some were more verbose, e.g., multi-page explanations of physiological processes. Most of the texts also contained information about other aspects of botany, e.g., experimental methods and historical developments; these were omitted from the analysis. We manually "parsed" each passage into an informal language of structure, function, and process which is commonly found in the discourse literature, e.g., (Mann and Thompson, 1987; McKeown, 1985; Paris, 1988; Souther et al., 1989; Suthers, 1988). Our final step was to generalize the most commonly occurring patterns into abstractions that covered as many aspects of the passages as possible. After generalizing the commonly occurring patterns into abstractions, we encoded the abstractions in two Explanation Design Packages. While this work was essential for gaining insights about biological texts, it was a sketchy and preliminary effort to informally characterize their content and organization. A promising line of future work is to construct a large corpus of "parsed" discourse through a formal analysis. This will enable the natural language generation community to begin making inroads into producing discourse in the same manner that corpus-based techniques have aided discourse understanding efforts.

The EDPs resulting from the analysis, *Explain-Process* and *Explain-Object*, can be used by an explanation planner to generate explanations about the processes and objects of physical systems. While these EDPs enable an explanation planner to generate quality explanations, we conjecture that employing a large library of specialized EDPs would produce explanations of higher quality. For the same reason that Kittredge *et al* note that domain-dependent discourse knowledge is critical for special purpose discourse generation (Kittredge, Korelsky, and Rambow, 1991), it appears that including EDPs specific to describing particular classes of biological processes, e.g., development and reproduction, would yield explanations whose content and organization better mirror that of explanations produced by domain experts.[8]

Although we will not discuss the details of the EDPs here, it is instructive to examine their structure and function. The *Explain-Process* EDP (Figure 5) can be used by the explanation planner to generate explanations about the processes that physical objects engage in. For example, given a query about how a biological process such as Embryo Sac Formation is carried out, the explanation planner can apply the *Explain-Process* EDP to construct an explanation plan that houses the content and organization of the explanation. The *Explain-Process* EDP has four primary topics

- *Process Overview*: Explains how a process fits into a taxonomy, discusses the role played by its actors, and discusses where it occurs;
- *Process Details*: Explains the steps of a process;
- *Temporal Attributes*: Explains how a process is related temporally to other processes;
- *Output Actor Fates*: Discusses how the "products" of a process are used by other processes.

As computational linguists have known for many years, formally characterizing texts is a very difficult, time-consuming, and error-prone process. Because any initial discourse representation effort by necessity must be considered only a beginning, the next step was to incrementally revise the EDPs. The EDPs were used to automatically construct hundreds of explanations: the explanation planner used the EDPs to construct explanation plans, and the realization system translated these plans to natural language.

The resulting explanations were presented to our domain expert, who critiqued both their content and organization, and we used these critiques to incrementally revise the EDPs. The majority of revisions involved the reorganization and removal of nodes in the EDPs. For example, the domain expert consistently preferred a different global organization than the one encoded in the original *Explain-Process* EDP. He also preferred explanations produced by a version of the *Explain-Process* EDP in which the information that had previously been associated with a *Process Significance* topic was associated with the *Temporal Attributes* topic. Moreover, he found that an *Actor Elaborations* node produced information that was "intrusive." Some revisions involved modifications to particular attributes of the nodes. For example, the inclusion condition on the original *Output Actor Fates* topic was TRUE. Instead, the domain expert preferred for explanations to include the content associated with this topic only when the process being described was a "conversion" process. After approximately twenty passes through the critiquing and revision phases, EDPs were devised that produced clear explanations meeting with the domain expert's approval.

5. Planning Explanations

Explanation planning is the task of determining the content and organization of explanations. We have designed an architecture for explanation generation and implemented a full-scale explanation generator, KNIGHT,[9] that is based upon this architecture.

8 While we have not explored this hypothesis in the work described here, the EDP framework can be used to test it empirically.

9 All of the explanation planning algorithms, as well as the KB Accessors, were implemented in Lucid Common Lisp on a DEC Station 5000.

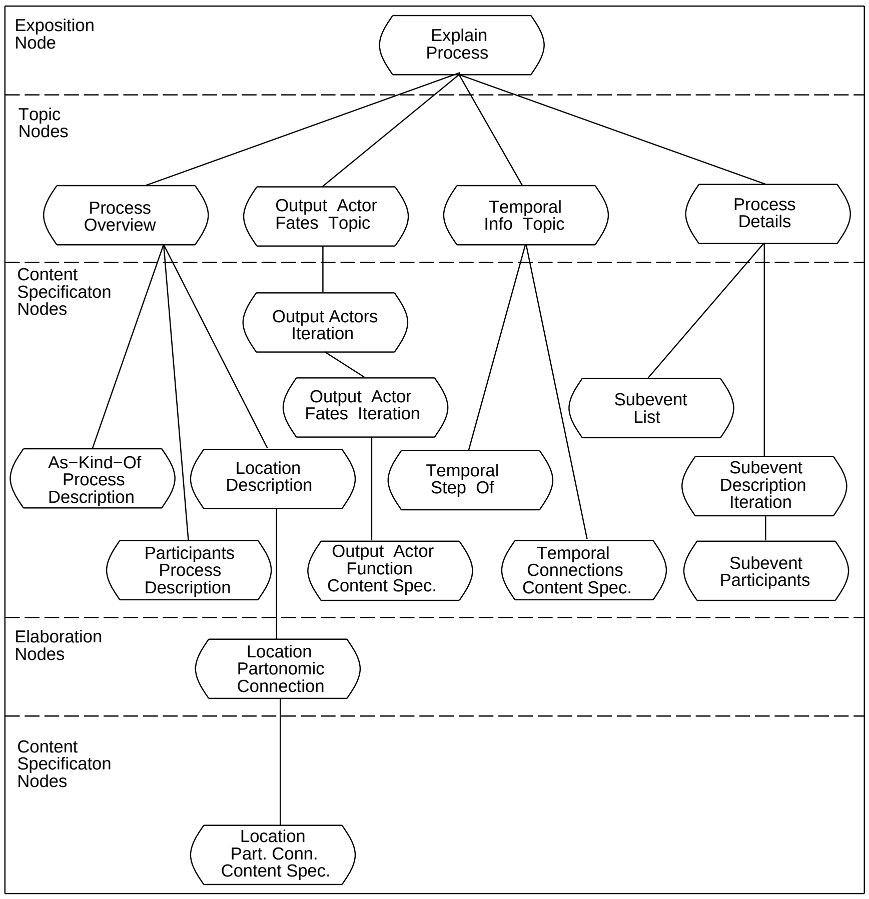

Exposition Node
Explain Process
Topic Nodes
Process Overview
Output Actor Fates Topic
Temporal Info Topic
Process Details
Content Specificaton Nodes
Output Actors Iteration
Output Actor Fates Iteration
Subevent List
As−Kind−Of Process Description
Location Description
Temporal Step Of
Subevent Description Iteration
Participants Process Description
Output Actor Function Content Spec.
Temporal Connections Content Spec.
Subevent Participants
Elaboration Nodes
Location Partonomic Connection
Content Specificaton Nodes
Location Part. Conn. Content Spec.

Figure 5
The Final Version of the Explain-Process Explanation Design

5.1 An Architecture for Explanation Generation

Explanation generation begins when the user poses a query, which includes a verbosity specification that comes in the form of a qualitative rating expressing the desired length of the explanation (Figure 6). The query interpreter—whose capabilities have been addressed only minimally in our work—translates the query to a canonical form, which is passed, along with the verbosity specification, to the explanation planner. Explanation planning is a synthetic task in which multiple resources are consulted to assemble data structures that specify the content and organization of explanations. KNIGHT's explanation planner uses the following resources: the Biology Knowledge Base, Explanation Design Packages, the KB accessing system, and an overlay user model.[10]

The explanation planner invokes the EDP Selector, which chooses an Explanation Design Package from the EDP Library. The explanation planner then applies the EDP by traversing its hierarchical structure. For each node in the EDP, the planner determines if it should construct a counterpart node in the explanation plan it is building. (Recall that the topic nodes and elaboration nodes of an EDP are instantiated only when their

10 As the planner constructs explanation plans, it consults an overlay user model (Carr and Goldstein, 1977). KNIGHT's user-sensitive explanation generation is not addressed in this paper. For a discussion of this work, see (Lester and Porter, 1991b; Lester, 1994).

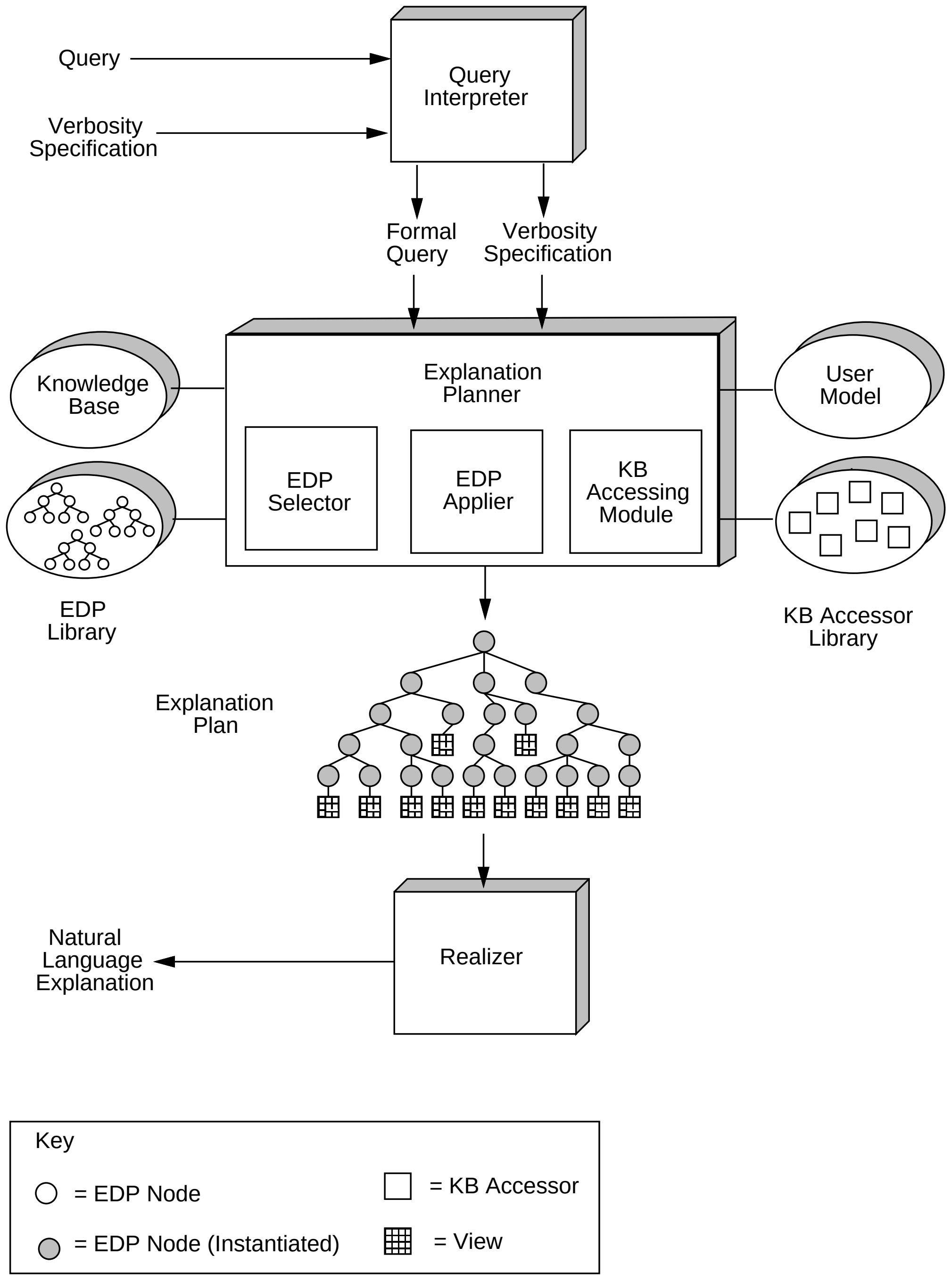


Figure 6
An Architecture for Explanation Generation

```
EXPLAIN ($Query\text{-}Type$, $Concept$, $Verbosity$)

if legal-query ($Query\text{-}Type$, $Concept$, $Verbosity$) then
    $EDP \leftarrow$ select-edp ($Query\text{-}Type$)
    $EDP\text{-}Exposition\text{-}Node \leftarrow$ get-root ($EDP$)
    $New\text{-}Exposition\text{-}Node \leftarrow$ construct-node ($EDP\text{-}Exposition\text{-}Node$)
    $Explanation\text{-}Plan \leftarrow$ apply-edp ($EDP\text{-}Exposition\text{-}Node$,
                                         $New\text{-}Exposition\text{-}Node$, $Verbosity$, nil)
    $Explanation\text{-}Leaves \leftarrow$ linearize ($Explanation\text{-}Plan$)
    realize ($Explanation\text{-}Leaves$)
```

Figure 7
The EXPLAIN Algorithm

conditions are satisfied.) As the plan is constructed, the explanation planner updates the user model to reflect the contextual changes that will result from explaining the views in the explanation plan, attends to the verbosity specification, and invokes KB Accessors to extract information from the knowledge base. Recall that the Accessors return "views," which are subgraphs of the knowledge base. The planner attaches the views to the explanation plan; they become the plan's leaves. Planning is complete when the explanation planner has traversed the entire EDP.

The planner passes the resulting explanation plan to the realization component (Section 6) for translation to natural language. The views in the explanation plan are grouped into *paragraph clusters*. After some "semantic polishing" to improve the content for linguistic purposes, the realization component translates the views in the explanation plan to sentences. The realization system collects into a paragraph all of the sentences produced by the views in a particular paragraph cluster. Explanation generation terminates when the realization component has translated all of the views in the explanation plan to natural language.

5.2 The Explanation Planning Algorithms

The EXPLAIN algorithm (Figure 7) is supplied with a query type (e.g., `Describe-Process`), a primary concept (e.g., `Embryo-Sac-Formation`), and a verbosity specification (e.g., `High`). Its first step is to select an appropriate EDP. The EDP Library has an indexing structure that maps a query type to the EDP that can be used to generate explanations for queries of that type. This indexing structure permits EDP selection to be reduced to a simple look-up operation. For example, given the query type *Describe Process*, the EDP Selector will return the *Explain-Process* Explanation Design Package. The planner is now in a position to apply the selected EDP to the knowledge base. The APPLY-EDP algorithm takes four arguments: the exposition node of the EDP that will be applied, a newly created exposition node which will become the root of the explanation plan that will be constructed, the verbosity specification, and the loop variable bindings.[11]

The planner first locates the root of the selected EDP, which is an exposition node. Next, it creates the corresponding exposition node for the soon-to-be-constructed explanation plan. It then invokes the APPLY EDP algorithm, which is given the exposition node of the EDP to be applied, the newly created exposition node that will become the root of the explanation plan, the verbosity, and a list of the loop variable bindings.[12]

11 APPLY EDP is a recursive algorithm. For top-level invocations, this latter parameter will always be `nil`.
12 The loop variable bindings are used for the EDPs' iteration construct.

```
APPLY-EDP (EDP-Exposit-Node, New-Exposit-Node, Verbosity,
           Loop-Var-Bindings)

Children-of-EDP-Exposition-Node
        ← get-children (EDP-Exposition-Node)
for each EDP-Topic-Node in EDP-Exposition-Node-Children do
    New-Topic-Node ← construct-node (EDP-Topic-Node)
    Inclusion-Condition-Expression ← get-condition (EDP-Topic-Node)
    Instantiated-Inclusion-Condition
              ← instantiate (Inclusion-Condition-Expression,
                             EDP-Topic-Node,
                             New-Topic-Node)
    Inclusion-Condition-Evaluation
           ← eval (Instantiated-Inclusion-Condition)
    Centrality ← get-centrality (EDP-Topic-Node)
    Include-Topic? ← compute-inclusion (Inclusion-Condition-Evaluation,
                                        Centrality,
                                        Verbosity)
    if Include-Topic? then
      Children-of-EDP-Topic-Node ← get-children (EDP-Topic-Node)
      for each EDP-Content-Specification-Node
          in Children-of-EDP-Topic-Node do
          determine-content (EDP-Content-Specification-Node,
                             New-Topic-Node,
                             Verbosity,
                             Loop-Var-Bindings)
```

Figure 8
The EDP Application Algorithm

The Apply EDP algorithm (Figure 8) and the algorithms it invokes traverse the hierarchical structure of the EDP to build an explanation plan. Its first action is to obtain the children of the EDP's exposition node; these are the topic nodes of the EDP. For each topic node, the EDP Applier constructs a new (corresponding) topic node for the evolving explanation plan. The Applier must then weigh several factors in its decision about whether to include the topic in the explanation: *inclusion*, which is the inclusion condition associated with the topic; *centrality*, which is the centrality rating that the discourse-knowledge engineer has assigned to the topic; and *verbosity*, which is the verbosity specification supplied by the user.

If the inclusion condition evaluates to False, the topic should be excluded regardless of the other two factors. Otherwise, the Compute Inclusion algorithm must consider the topic's importance and the amount of detail requested and will include the topic in the following circumstances: the verbosity is `High`; the verbosity is `Low` but the topic's centrality has been rated as `High` by the discourse knowledge engineer; or the verbosity is `Medium` and the topic's centrality has been rated as `Medium` or `High`.

When the Compute Inclusion algorithm returns True, the Applier obtains the children of the EDP's topic. These are its content specification nodes. For each of the topic's content specification nodes, the Applier invokes the Determine Content algorithm, which itself invokes KB Accessors named in the EDP's content specification nodes. This action extracts views from the knowledge base and attaches them to the explanation plan.

To determine the content of the information associated with elaboration nodes, Determine Content invokes the Apply EDP algorithm. Because it was the Apply EDP algorithm that invoked Determine Content, this is a recursive call. In this invocation of Apply EDP—as opposed to the "top-level" invocation by the Explain algorithm—Apply EDP is given an elaboration node instead of a topic node. By recursively invoking Apply EDP, Determine Content causes the planner to traverse the elaboration branches of a content node. The recursion bottoms out when the system encounters the leaves of the EDP, i.e., content specification nodes in the EDP that do not have elaborations.

Rather than merely returning a flat list of views, the Explain algorithm examines the paragraph specifications in the nodes of the EDP it applied. The paragraph specifications of a given node organize the children of that node into *paragraph clusters*. The order of the paragraph clusters controls the global structure of the final textual explanation; the order of the views in each paragraph cluster determines the order of sentences in the final text.[13] Finally, the Explain algorithm passes the paragraph clusters to the Realize algorithm, which translates them to natural language.

6. Realization

The explanation planner should be viewed as an automatic specification writer: its task is to write specifications for the realization component, which interprets the specifications to produce natural language. Although our work focuses on the design, construction, and evaluation of explanation planners, by constructing a full-scale natural language generator, it becomes possible to conduct a "pure" empirical evaluation of explanation planners. Without a realization component, the plans produced by an explanation planner would need to be manually translated to natural language, which would raise questions about the purity of the experiments. We therefore designed and implemented a full-scale realization component.[14]

Realization can be decomposed into two subtasks: *functional realization*, constructing functional descriptions from message specifications supplied by a planner; and *surface generation*, translating functional descriptions to text. *Functional descriptions* encode both semantic information (case assignments) and structural information (phrasal constituent embeddings). Syntactically, a functional description is a set of attribute and value pairs `(a v)` (collectively called a *feature set*), where `a` is an attribute (a feature) and `v` is either an atomic value or a nested feature set.[15] To illustrate, Figure 9 depicts a sample functional description. The first line, `(cat clause)`, indicates that what follows will be some type of verbal phrase, in this case a sentence. The second line contains the keyword `proc`, which denotes that everything in its scope will describe the structure of the entire verbal phrase. The next structure comes under the heading `partic`; this is where the thematic roles of the clause are specified. In this instance, one thematic role exists in the main sentence, the `agent` (or subject), which is further defined by its lexical entry and a modifying prepositional phrase indicated by the keyword `qualifier`. The structure beginning with `circum` creates the subordinate infinitival purpose clause. It has two thematic roles, subject and object. The subject has a pointer to identify itself with the subject of the main clause while the object contains a typical noun phrase. The feature set for the `circum` clause indicates the wide range of possibilities for placement

13 The realization algorithm treats these groupings as suggestions which may be overridden in extenuating circumstances.

14 During the past few years, we have developed a series of realization systems. The first realizer, which was designed and implemented by the first author, was a template-based generator. The second realizer, which was designed by Kathy Mitchell and the authors (Mitchell, 1992), used the Penman (Mann, 1983) surface generator. The third realizer (Callaway and Lester, 1995) is described briefly in this section; it was developed by the first author and Charles Callaway.

15 Functional descriptions may also employ syntactic sugar for purposes of legibility.

```
((cat clause)
 (proc ((type material) (lex "reproduce")))
 (partic ((agent ((cat common)
                  (lex "spore")
                  (qualifier ((cat pp)
                              (prep === "from")
                              (np ((cat common)
                                   (lex "cell")
                                   (classifier ((cat noun-compound)
                                                (classifier === "megaspore")
                                                (head === "mother")))
                                   (qualifier ((cat pp)
                                               (prep === "in")
                                               (np ((cat common)
                                                    (lex "sporangium")
                                                    )))))))))))
 (circum ((purpose ((cat clause) (position end)
                    (keep-for no) (keep-in-order no)
                    (proc ((type material) (lex "form")))
                    (partic ((agent ((semantics (partic agent semantics))))
                             (affected ((cat common)
                                        (lex "gamete") (definite no)
                                        (classifier === "plant")
                                        (describer === "haploid")
                                        (cardinal ((value 4) (digit no))))
                                       )))))
          (time ((time-type "during")
                 (cat common)
                 (describer === "male")
                 (classifier === "gametophyte")
                 (lex "generation"))))))
```

Figure 9
A Functional Description

of the clause as well as for introducing additional phrasal substructures into the purpose clause.

To construct functional descriptions from views extracted from a knowledge base, KNIGHT employs a functional realization system (Callaway and Lester, 1995). Given a view, the functional realizer uses its knowledge of case mappings, syntax, and lexical information to construct a functional description, which it then passes to the FUF surface generator. The functional realizer consists of five principal components:

- *Lexicon:* Physically distributed throughout the knowledge base; each concept frame has access to all of the lexical information relevant to its own realization.
- *Functional Description Skeleton Library:* Contains a large number of FD-Skeletons, each of which encodes the associated syntactic, semantic, and role assignments for interpreting a specific type of message specification.
- *Functional Description Skeleton Retriever:* Charged with the task of selecting the correct Functional Description Skeleton from the skeleton library.
- *Noun Phrase Generator:* Responsible for drawing lexical information from the Lexicon to create a self-contained functional description representing each noun phrase required by the FD-Skeleton processor
- *Functional Description Skeleton Processor:* Gathers all of the available information from the FD-Skeleton, the lexicon, and the noun phrase

generator; produces the final functional description.

When the functional realizer is given a view, its first task is to determine the appropriate FD-Skeleton to use. Once this is accomplished, the FD-Skeleton is passed along with the message specification to the FD-Skeleton processor. The FD-Skeleton processor first determines if each of the essential descriptors is present; if any of these tests fail, it will note the deficiency and abort. If the message is well-formed, the FD-Skeleton processor passes each realizable concept unit found on the message specification to the noun phrase generator, which uses the lexicon to create a functional description representing each concept unit. The noun phrase generator then returns each functional description to the FD-Skeleton processor, which assigns case roles to the (sub)-functional descriptions. The resulting functional description, which encodes the functional structure for the entire content of the message specification, is then passed to the surface realizer. Surface realization is accomplished by FUF (Elhadad, 1992). Developed by Elhadad and his colleagues at Columbia, FUF is accompanied by an extensive, portable English grammar, which is "the result of five years of intensive experimentation in grammar writing" and is currently the largest "generation grammar" in existence (Elhadad, 1992). Given a set of functional descriptions, FUF constructs the final text.

7. Example Behavior

To illustrate the behavior of the system, consider the concept of *embryo sac formation*. Figure 2 depicts the semantic network in the Biology Knowledge Base that represents information about *embryo sac formation*. When KNIGHT is given the task of explaining this concept,[16] it applies the *Explain-Process* EDP (Figure 5).

KNIGHT first finds the topics of the *Explain Process* exposition node, which are *Process Overview*, *Output Actor Fates*, *Temporal Information*, and *Process Details*. During its traversal of this tree, it begins with *Process Overview*, which has a `High` centrality rating and an inclusion condition of TRUE. KNIGHT executes the COMPUTE INCLUSION algorithm with the given verbosity of `High`, which returns TRUE, i.e., the information associated with the topic should be included.

Hence, it now begins to traverse the children of this topic node, which are the *As Kind Of Process Description*, *Process Participants*, and *Location Description* content specification nodes. For the *As Kind Of Process Description*, it computes a value for the local variable *?Reference Concept*, which returns the value *female gametophyte formation*. It then instantiates the content specification template on *As Kind Of Process Description*, which it then evaluates. This results in a call to the *As Kind Of* KB Accessor, which produces a view. The view produced in this execution will eventually be translated to the sentence, "Embryo sac formation is a kind of female gametophyte formation." Similarly, KNIGHT instantiates the content specification expressions of *Process Participants Description* and *Location Description*, which also cause KB Accessors to be invoked; these also return views. The first of these views will be used to produce the sentence, "During embryo sac formation, the embryo sac is formed from the megaspore mother cell." and the second will produce the sentence, "Embryo sac formation occurs in the ovule." Next KNIGHT visits the *Location Partonomic Connection* node, which is an elaboration of *Location Description*. However, because its inclusion condition is not satisfied, this branch of the traversal halts.

16 In this example, KNIGHT is given a HIGH verbosity specification. Details of this example, as well as other examples, may be found in Chapter 4 of (Lester, 1994).

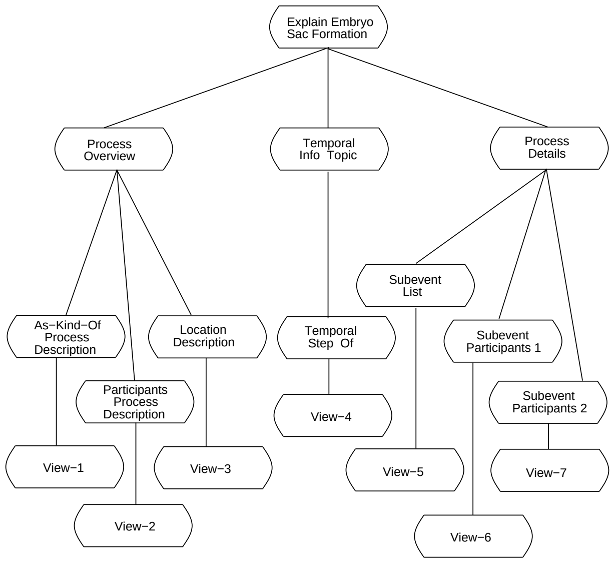


Figure 10
An explanation plan for embryo sac formation: High verbosity

Next, KNIGHT visits each of the other topics of the *Explain Process* exposition node: *Output Actor Fates*, *Temporal Information* and *Process Details*. When it visits the *Output Actor Fates* topic, the inclusion condition is not satisfied. Because it was given a `High` verbosity specification and the inclusion conditions are satisfied, both *Temporal Information* and *Process Details* are used to determine additional content. The view constructed from *Temporal Information* will produce the sentence, "Embryo sac formation is a step of angiosperm sexual reproduction." and the *Process Details* will result in the generation of descriptions of the steps of embryo sac formation, namely, megasporogenesis and embryo sac generation. When the views in the resulting explanation plan (Figure 10) are translated to text by the realization system, KNIGHT produces the explanation shown in Figure 1.

These algorithms have been used to generate explanations about hundreds of different concepts in the Biology Knowledge Base. For example, Section 2 shows other explanations generated by KNIGHT. The explanation of *pollen tube growth* was produced by applying the *Explain-Process* EDP, and the explanations of *spore* and *root system* were produced by applying the *Explain-Object* EDP.

8. Evaluation

Traditionally, research projects in explanation generation have not included empirical evaluations. Conducting a formal study with a generator has posed difficulties for at least three reasons: the absence of large-scale knowledge bases; the problem of robustness; and the subjective nature of the task. First, the field of explanation generation has experienced a dearth of "raw materials." The task of an explanation generator is threefold: to extract information from a knowledge base, to organize this information, and to translate it to natural language. Unless an explanation generator has access to a sufficiently large knowledge base, the first step—and hence the second and third—cannot be carried out enough times to evaluate the system empirically. Unfortunately, because of the tremendous cost of construction, large-scale knowledge bases are scarce.

Second, even if large-scale knowledge bases were more plentiful, an explanation generator cannot be evaluated unless it is sufficiently robust to produce many explanations. In very practical terms, a generator is likely to halt abruptly when it encounters unusual and unexpected knowledge structures; if this happens frequently, the system will generate too few explanations to enable a meaningful evaluation. We conjecture that most implemented explanation generators would meet with serious difficulties when applied to a large-scale knowledge base.

Third, explanation generation is an ill-defined task. It stands in contrast to a machine learning task such as rule induction from examples. Although one can easily count the number of examples that an induction program classifies correctly, there is no corresponding objective metric for an explanation generator. Ideally, we would like to "measure" the coherence of explanations. Although it is clear that coherence is of paramount importance for explanation generation, there is no litmus test for it.

Given these difficulties, how can one evaluate the architectures, algorithms, and knowledge structures that form the basis for an explanation generator? The traditional approach has been to conduct an analytical evaluation of a system's architecture and demonstrate that it can produce well-formed explanations on a few examples. While this evaluation technique is important, it is not sufficient. Three steps can be taken to promote better evaluation. First, we can construct large-scale knowledge bases, such as the Biology Knowledge Base. Second, we can design and implement robust explanation systems that employ a representation of discourse knowledge that is easily manipulable by discourse-knowledge engineers. Third, to ensure that a knowledge base is not tailored for the purposes of explanation generation, we can enter into a "contractual agreement" with knowledge engineers; this eliminates all requests for representational modifications that would skew the representation to the task of explanation generation.

8.1 Experimental Design

The **Two-Panel Evaluation Methodology** can be used to empirically evaluate natural language generation work. We developed this methodology, which involves two panels of domain experts, to combat the inherent subjectivity of NLG: although multiple judges will rarely reach a consensus, their collective opinion provides persuasive evidence about the quality of explanations. To ensure the integrity of the evaluation results, a central stipulation of the methodology is that the following condition be maintained throughout the study:

> **Computer Blindness:** None of the participants can be aware that some texts are machine-generated or, for that matter, that a computer is in any way involved in the study.

The methodology involves four steps:

1. Generation of explanations by computer.
2. Formation of two panels of domain experts.
3. Generation of explanations by one panel of domain experts.
4. Evaluation of all explanations by second panel of domain experts.

Each of these is discussed in turn.

Explanation Generation: Knight. Because KNIGHT's operation is initiated when a user poses a question, the first task was to select the questions it would be asked. To this end, we combed the Biology Knowledge Base for concepts that could furnish topics for questions. Although the knowledge base focuses on botanical anatomy, physiology, and development, it also contains a substantial amount of information about biological taxons. Because this latter area is significantly less developed, we ruled out concepts about taxons. In addition, we ruled out concepts that were too abstract, e.g., *Object*. We then requested KNIGHT to generate explanations about the 388 concepts that passed through these filters.

To thoroughly exercise KNIGHT's organizational abilities, we were most interested in observing its performance on longer explanations. Hence, we eliminated explanations of concepts that were sparsely represented in the knowledge base. To this end, we passed the 388 explanations through a "length filter": explanations that consisted of at least 3 sentences were retained; shorter explanations were disposed of.[17] This produced 87 explanations, of which 48 described objects and 39 described processes. Finally, to test an equal number of objects and processes, we randomly chose 30 objects and 30 process.

Two Panels of Domain Experts. To address the difficult problem of subjectivity, we assembled 12 domain experts, all of whom were PhD students or post-doctoral scientists in biology. Because we wanted to gauge KNIGHT's performance relative to humans, we assigned each of the experts to one of two panels: the *Writing Panel* and the *Judging Panel*. By securing the services of such a large number of domain experts, we were able to form relatively large panels of four writers and eight judges (Figure 11). To promote high quality human-generated explanations, we assigned the four most experienced experts to the Writing Panel. The remaining eight experts were assigned to the Judging Panel to evaluate explanations.

To minimize the effect of factors that might make it difficult for judges to compare KNIGHT's explanations with those of domain experts, we took three precautions. First, we attempted to control for the length of explanations. Although we could not impose hard constraints, we made suggestions about how long a typical explanation might be. Second, to make the "level" of the explanations comparable, we asked writers to compose explanations for a particular audience, freshman biology students. Third, so that the general topics of discussion would be comparable, we asked writers to focus on anatomy, physiology, and development.

Explanation Generation: Humans. To ensure that the difficulty of the concepts assigned to the writers were the same as those assigned to KNIGHT, the writers were given the task of explaining *exactly* the same set of concepts that KNIGHT had explained. Because we wanted to give writers an opportunity to explain both objects and processes, each writer was given an approximately equal number of objects and processes. Each of the 4 writers was given 15 concepts to explain, and each concept was assigned to exactly one writer. We then transcribed their handwritten explanations and put them and KNIGHT's explanations into an identical format. At this point, we had a pool of 120 explanations: sixty of these pertained to objects (30 written by biologists and 30 by KNIGHT), and the other sixty pertained to processes (also 30 written by biologists and 30 by KNIGHT).

Explanation Evaluation. We then submitted the explanations to the panel of eight judges. The judges were not informed of the source of the explanations, and all of the explanations appeared in the same format. Each judge was given fifteen explanations to evaluate.

17 A separate study would be to evaluate KNIGHT on very short (one-sentence and two-sentence) explanations. However, this study would be an evaluation of how it behaves in the face of highly incomplete knowledge rather than a fair head-to-head comparison with knowledgeable experts.

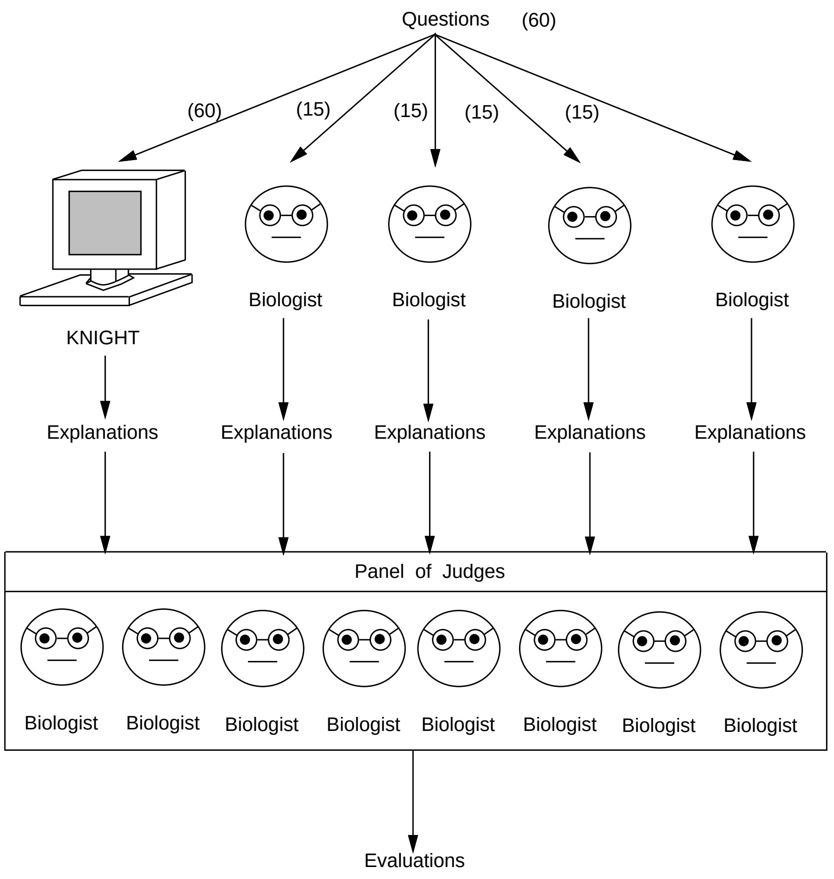


Figure 11
The Two-Panel Methodology in the Knight Experiments

Judges were asked to rate the explanations on several dimensions: overall quality and coherence, content, organization, writing style, and correctness. To provide judges with a familiar rating scale, they were asked to assign letters grades (A, B, C, D, or F) to each explanation on each of the dimensions. Because carefully evaluating multiple dimensions of explanations is a labor-intensive task, time considerations required us to limit the number of explanations submitted to each judge. Hence, we assigned each judge a set of 15 explanations. (On average, each judge took an hour to evaluate 15 explanations.) We assigned explanations to judges using an allocation policy that obeyed the following four constraints:

- *System-Human Division:* Each judge received explanations that were approximately evenly divided between those that were produced by Knight and those that were produced by biologists.
- *Object-Process Division:* Each judge received explanations that were approximately evenly divided between objects and processes.

- *Single-Explanation Restriction:* No judge received two explanations of the same concept.[18]
- *Multi-Judge Stipulation:* The explanations written by each writer were parceled out to at least two judges, i.e., rather than having one judge evaluate one writer's explanations, that writer's explanations were distributed among multiple judges.

It is important to emphasize again that the judges were not made aware of the purpose of the experiment, nor were they told that any of the explanations were computer-generated.

8.2 Results

By the end of the study, we had amassed a large volume of data. To analyze it, we converted each of the "grades" to their traditional numerical counterparts, i.e., A=4, B=3, C=2, D=1, and F=0. Next, we computed means and standard errors for both KNIGHT's and the biologists' grades. We calculated these values for the overall quality and coherence rating, as well as for each of the dimensions of content, organization, writing style, and correctness. On the overall rating and on each of the dimensions, KNIGHT scored within approximately "half a grade" of the biologists (Table 3).[19]

Given these results, we decided to investigate the differences between KNIGHT's grades and the biologists' grades. When we normalized the grades by defining an "A" to be the mean of the biologists' grades, KNIGHT earned approximately 3.5 (a B^+). Comparing differences in dimensions, KNIGHT performed best on correctness and content, not quite as well on writing style, and least well on organization.

Because the differences between KNIGHT and the biologists were narrow in some cases, we measured the statistical significance of these differences by running standard t-tests.[20] KNIGHT's grades on the content, organization, and correctness dimensions did not differ significantly from the biologists' (Table 4). Of course, an insignificant difference does not indicate that KNIGHT's performance and the biologists' performance was equivalent—an even larger sample size might have shown a significant difference—however, it serves as an indicator that KNIGHT's performance approaches that of the biologists on these three dimensions.

To gauge how well KNIGHT generates explanations about objects—as opposed to processes—we computed means and standard errors for both KNIGHT's explanations of objects and the biologists' explanations of objects. We did the same for the explanations of processes. For both objects and processes, KNIGHT scored within "half a grade" of the biologists. Again, we measured the statistical significance of these differences. Although there was a significant difference between KNIGHT and biologists on explanations of processes, KNIGHT and the biologists did not differ significantly on explanations of objects (Tables 5 and 6). A probable cause of this result lies in the domain: in biology, process explanations are often more complex than object explanations, therefore making process explanations more challenging to generate.

As a final test, we compared KNIGHT to each of the individual writers. For a given writer, we assessed KNIGHT's performance relative to that writer in the following way: we compared the grades awarded to KNIGHT and the grades awarded to the writer on explanations generated in response to the same set of questions. This analysis produced

18 The purpose of this constraint is to promote immediate, non-deliberative reactions from the judges. An alternate study would consist of judges consciously analyzing pairs of explanations to perform an explicit comparative analysis.
19 In the tables, ± denotes the standard error, i.e., the standard deviation of the mean.
20 All t-tests were unpaired, two-tailed. The results are reported for a 0.05 level of confidence.

some surprising results. Although there were substantial differences between KNIGHT and "Writer 1," KNIGHT was somewhat closer to "Writer 2," it was very close to "Writer 3," and its performance actually exceeded that of "Writer 4." KNIGHT and Writers 2, 3, and 4 did not differ significantly (Table 7).

9. Related Work

By synthesizing a broad range of research in natural language generation, KNIGHT provides a "start-to-finish" solution to the problem of automatically constructing expository explanations from semantically rich, large-scale knowledge bases. It introduces a new evaluation methodology and builds on the conceptual framework that has evolved in the NLG community over the past decade, particularly in techniques for knowledge-base access and discourse knowledge representation. We discuss each of these in turn.

Evaluation Methodologies. With regard to evaluation, KNIGHT is perhaps most closely related to five NLG projects that have been empirically evaluated: PAULINE (Hovy, 1990), EDGE (Cawsey, 1992), the Example Generator[21] (Mittal, 1993), ANA (Kukich, 1983), and STREAK (Robin, 1994). By varying pragmatic information such as tone, Hovy enabled PAULINE to generate many different paragraphs on the same topic. PAULINE's texts were not formally analyzed by a panel of judges, and it did not produce texts on a wide range of topics (it generated texts on only three different events). However, this project is a significant achievement in terms of evaluation *scale* because of the sheer number of texts it produced: PAULINE generated more than 100 different paragraphs on the same subject. In a second landmark evaluation, Cawsey undertook a study in which subjects were allowed to interact with her explanation generation system, EDGE (Cawsey, 1992). Subjects posed questions to EDGE about the operation of four circuits. Cawsey analyzed the system's behavior as the dialogs progressed, interviewed subjects, and used the results to revise the system. Although EDGE does not include a realization system (other than simple templates) and it was not subjected to a tightly controlled, formal evaluation, it was sufficiently robust to be used interactively by eight subjects.

The EXAMPLE GENERATOR (Mittal, 1993), ANA (Kukich, 1983), and STREAK (Robin, 1994) were each subjected to formal (quantitative) evaluations. Mittal and Paris developed and formally evaluated a generator that produced descriptions integrating text and examples. Rather than evaluating the explanations directly, subjects were given a quiz about the concept under consideration.[22] The degree to which the experiments controlled for specific factors, e.g., the effect of example positioning, example types, example complexity, and example order, is remarkable. ANA and STREAK were both subjected to quantitative, corpus-based evaluations. Kukich employed a corpus-based methodology to judge the coverage of ANA's knowledge structures. STREAK, which constructs summaries of basketball games, is part of of a larger effort by J. Robin, K. McKeown, and their colleagues at Columbia and Bellcore to develop robust document generation systems (McKeown, Robin, and Kukich, 1995). It was evaluated with a corpus-based study that produced estimates of STREAK's sub-language coverage, extensibility, and the overall effectiveness of its revision-based generation techniques. Although neither of these studies employed human judges to critique text quality, the rigor with which they were conducted has significantly raised the standards for evaluating generation systems.

21 Mittal and Paris' system has no official name; we refer to it as "the Example Generator" for ease of reference.

22 In a second analysis *without human judges*, the system developers compared selected features of the EXAMPLE GENERATOR's output with text from textbook and obtained encouraging results.

Table 3
Comprehensive Analysis

Generator	*Overall*	*Content*	*Organization*	*Writing*	*Correctness*
KNIGHT	2.37±0.13	2.65±0.13	2.45±0.16	2.40±0.13	3.07±0.15
Human	2.85±0.15	2.95±0.16	3.07±0.16	2.93±0.16	3.16±0.15

Table 4
Differences and Significance

	Overall	*Content*	*Organization*	*Writing*	*Correctness*
Difference	0.48	0.30	0.62	0.53	0.09
t statistic	-2.36	-1.47	-2.73	-2.54	-0.42
Significance	0.02	0.14	0.07	0.01	0.67
Significant?	Yes	No	No	Yes	No

Table 5
Explanation of Objects

Generator	*Grade*
KNIGHT	2.65±0.19
Human	2.93±0.19
Difference	0.28
t statistic	-1.05
Significance	0.30
Significant?	No

Table 6
Explanation of Processes

Generator	*Grade*
KNIGHT	2.10±0.24
Human	2.77±0.17
Difference	0.67
t statistic	-2.23
Significance	0.03
Significant?	Yes

Table 7
KNIGHT vs. Individual Writers

KNIGHT	*vs. Writer 1*	*vs. Writer 2*	*vs. Writer 3*	*vs. Writer 4*
KNIGHT	1.93±0.29	2.73±0.23	2.73±0.27	2.07±0.23
Human	3.60±0.16	3.40±0.23	2.80±0.28	1.60±0.23
Difference	1.67	0.67	0.07	0.47
t statistic	-5.16	-2.03	-0.17	1.42
Significance	0.00	0.05	0.86	0.16
Significant?	Yes	No	No	No

Table 8
Evaluation Methodologies

	PAULINE	EDGE	EXAMPLE GENERATOR	ANA	STREAK	KNIGHT
Formality	Informal	Informal	Formal	Formal	Formal	Formal
"Judges"	None	Humans	Humans	Corpus	Corpus	Humans
Large-Scale KB?	No	No	No	No	No	Yes
System *vs.* Human	No	No	Indirect	No	No	Yes

The relationship between the KNIGHT evaluation and those of its predecessors is summarized in Table 8. KNIGHT, STREAK, and ANA were all evaluated formally, i.e., quantitatively, while PAULINE and EDGE were evaluated informally. The KNIGHT, EDGE, and EXAMPLE GENERATOR evaluations employed humans as judges, while the ANA and STREAK evaluations had "artificial judges" in the form of corpora, and PAULINE was evaluated without judges. KNIGHT is the only system to have been evaluated in the context of a semantically rich, large-scale knowledge base. KNIGHT is also the only system to have been evaluated in a kind of restricted "Turing test" in which the quality of its text was evaluated by humans in a head-to-head comparison against the text produced by humans (domain experts) in response to the same set of questions.

Knowledge-Base Access. Several projects in explanation generation have exploited views to improve the quality of the explanations they provide. The ADVISOR system (McKeown, Wish, and Matthews, 1985) represents views with a multiple-hierarchy knowledge base. ADVISOR infers a user's current goal from his most recent utterances and uses this goal to select a hierarchy from the multiple-hierarchy knowledge base. The selected view controls the content of the explanation and the reasoning that produced that content. In a similar vein, viewpoints in Swartout's XPLAIN (Swartout, 1983) are annotations that indicate when to include a piece of knowledge in an explanation.

It is preferable to construct (i.e., extract) views at runtime rather than encoding them in a knowledge base. If a KB accessing system could dynamically construct views, the discourse-knowledge engineer would be freed from the task of having to anticipate all queries and rhetorical situations and precompiling semantic units for each situation. KNIGHT, ROMPER (McCoy, 1989 1990), and Suthers' work (Suthers, 1988; Suthers, 1993) use these types of views to determine the content of their explanations. Once a perspective is selected, ROMPER includes in its explanations only those attributes whose salience values are the highest. In contrast to ROMPER's views which are domain-specific, i.e., its views are confined to the domain of financial securities, Suthers' and KNIGHT's views are domain independent. Suthers set forth a set of views which can be used to select coherent subsets of domain knowledge: structural, functional, causal, constraint, and process. He also developed a view retriever and a highly refined theory of explanation generation in which views play a significant role. KNIGHT's views are very similar to McCoy's and Suthers' in that they define the relations and properties of a concept that are relevant when considering a concept from a viewpoint belonging to that view type (Acker, 1992; Souther et al., 1989). They also provide four types of knowledge-base access robustness, as discussed in Section 3.

Discourse Generation. Two principle mechanisms have been developed for generating discourse: schemata and top-down planners.[23] McKeown's pioneering work on *schemata* marks the beginning of the "modern era" of discourse generation (McKeown, 1985). Schemata are ATN-like structures that represent naturally occurring patterns of discourse. For example, a schema for defining a concept includes instructions to identify its superclass, to name its parts, and to list its attributes. To construct an explanation plan, McKeown's TEXT system traverses the schemata and sequentially instantiates rhetorical predicates with propositions from a knowledge base. Paris extended schemata to generate descriptions of complex objects in a manner that is appropriate for the user's level of expertise (Paris, 1988), and ROMPER's schemata include information about the content of propositions to be selected, as well as their communicative role. Although schemata have been criticized because they lack flexibility, they successfully capture many aspects of discourse structure.

An alternative to schemata is the top-down planning approach (Cawsey, 1992; Hovy, 1993; Maybury, 1992; Moore, 1995; Moore and Paris, 1993; Suthers, 1991).[24] The operators of two of these planning systems are based on Rhetorical Structure Theory (RST) (Mann and Thompson, 1987). Hovy's Structurer (Hovy, 1993) is a hierarchical planner whose operators instantiate relations from RST. The Reactive Planner also uses RST-like operators. However, unlike all of the preceding research—and unlike KNIGHT as well—it offers sophisticated mechanisms for generating explanations in interactive contexts (Moore, 1995; Moore and Paris, 1993). Because the operators explicitly record the rhetorical effects achieved, and because the system records alternative operators it could have chosen, as well as assumptions it made about the user, the Reactive Planner can respond to follow-up questions—even if they are ambiguous—in a principled manner. A related approach has been taken by Cawsey in the EDGE system (Cawsey, 1992). Because EDGE has facilities for managing conversations, users may interrupt the system to ask questions, and EDGE can either answer the question immediately or postpone its response. Suthers (Suthers, 1991) has developed a sophisticated hybrid approach that includes planning techniques as well as plan critics, simulation models, reorganization methods, and graph traversal. By assembling these diverse mechanisms into a single architecture, he demonstrates how the complexities of explanation planning can be dealt with in a coherent framework. The principal advantage of top-down planners over schema-based generators is their ability to reason about the structure, content, and

23 A third alternative proposed by Sibun are *short-range* strategies that exploit relations such as spatial proximity to guide the generator through the domain knowledge (Sibun, 1992). Though flexible, they do not account for extended explanations, which require a more global rhetorical structure.

24 The planning approach, which has dominated the field for the past few years can be traced to Appelt's work on planning referring expressions (Appelt, 1985), which itself builds on earlier work on reasoning about speech acts in a planning paradigm (Cohen and Perrault, 1979).

goals of explanations—as opposed to merely instantiating pre-existing plans embodied by schemata.

KNIGHT's EDPs are much more schema-like than plan-like. Although EDPs have inclusion conditions, which are similar to the *constraint* attribute of RST-based operators, and they provide a *centrality* attribute, which enables KNIGHT to reason about the inclusion of a topic if "space" is limited, EDPs do not in general permit KNIGHT to reason about the goals fulfilled by particular text segments as do plan-based systems. For example, if the expressions in an EDP's inclusion condition are not satisfied, KNIGHT cannot create a plan to satisfy them. Moreover, although EDPs are effective for generating explanations, developing EDPs to achieve other communicative goals, e.g., `Correct-Misconception`, may be beyond their capabilities. Despite these drawbacks, EDPs have proven to be very useful as a discourse-knowledge engineering tool, a result that can be attributed in large part to their combining a frame-based representation with procedural constructs. In a sense, EDPs are schemata whose representation has been fine-tuned to maximize ease of use on a large scale.

10. Future Directions

Research on KNIGHT suggests several directions for future work. First, the results of the evaluation call for further analysis and experimentation. For example, an in-depth analysis at the discourse, sentential and lexical levels of all of the texts produced by both the humans and the system may reveal which characteristics of the highly rated texts are desirable. These in turn can be used to improve the EDPs. On the empirical side, a particularly intriguing kind of experiment is an *ablation* study. In an ablation study, different aspects of the system are ablated (removed or degraded), and the effects of the ablation on the explanations are noted. By performing a series of these experiments, one can determine which aspects of the system and its representational structures contribute most significantly to its success.

Second, although EDPs were employed successfully in the generation of hundreds of explanations, the fact that they have more in common with schemata than with the operators of top-down planners is indicative of a fundamental limitation: the intentional structure of the discourse is unavailable for inference. As a result, it is considerably more difficult for the system to respond to follow-up questions, reason about paragraph structure, perform goal-based content determination, and produce discourse cues. This calls for the incorporation of an intentional structure into EDPs, but modifying EDPs to represent intention must be accomplished in such a way that the "discourse-knowledge engineering" properties are preserved and there is no sacrifice of text quality. Moreover, KNIGHT currently employs very rudimentary pronominalization techniques. Including more sophisticated methods (Dale, 1992), perhaps in concert with an intentional structure, should result in a significant increase in text quality.

Third, the issue of portability is of considerable interest. Porting to another domain and/or task will involve three steps. First, after the domain knowledge has been represented, a discourse-knowledge engineer will develop EDPs that are appropriate for the new domain and task. Second, the realization system must be extended by adding new functional description skeletons. Third, the lexicon must be created. It is our hypothesis that because EDPs are designed for ease of creation and modification, the second and third steps will be considerably more difficult than the first. Empirically exploring this hypothesis presents an interesting line of future work.

Finally, one of the most fruitful areas for future work is research on animated explanation generation. To this end, we have begun work on an animated pedagogical agent (Stone and Lester, 1996) that can explain complex concepts with dynamically sequenced visual and verbal behaviors. These kinds of systems must be able to address issues of animated content determination and organization, as well as determining at runtime which media should be used to realize the concepts to be communicated.

11. Conclusion

Explanation generation is an exceedingly complex task that involves a diversity of interacting computational mechanisms. An explanation system must be able to select from a knowledge base precisely those facts that enable it to construct a response to a user's question, organize this information and translate the formal representational structures found in knowledge bases to natural language. While the traditional approach to work on explanation has been to develop a proof-of-concept system and to demonstrate that it can produce well-formed explanations on a few examples, developing robust explanation generation techniques and scalable discourse knowledge representations facilitates more extensive, empirical studies.

To investigate the issues and problems of generating natural language explanations from semantically rich, large-scale knowledge bases, we have designed and implemented KNIGHT, a fully functioning explanation system that automatically constructs multi-sentential and multi-paragraph natural language explanations. KNIGHT has generated hundreds of explanations from the Biology Knowledge Base. It addresses a multiplicity of issues in explanation generation, ranging from knowledge base access and discourse planning to a new methodology for empirical evaluation. This work has demonstrated that (1) separating out knowledge-base access from explanation planning can enable the construction of a robust system that extracts coherent views from a a semantically rich, large-scale knowledge base; and (2) Explanation Design Packages, a hybrid representation of discourse knowledge that combines a frame-based representation with procedural constructs, facilitate the iterative refinement of discourse knowledge. Combining hierarchically structured discourse objects with embedded procedural constructs, EDPs have been used to represent discourse knowledge about explaining physical objects and processes, and they have been tested in the generation of hundreds of explanations of biological concepts.

To gauge the effectiveness of these techniques, we developed the Two-Panel Evaluation Methodology and employed it in the evaluation of KNIGHT. KNIGHT scored within "half a grade" of the biologists. There was no significant difference between KNIGHT's explanations and the biologists' explanations on measures of content, organization, and correctness, nor was there a statistically significant difference in overall quality between KNIGHT's explanations and those composed by three of the biologists. KNIGHT's performance exceeded that of one of the biologists.

In summary, it is encouraging that an explanation system could begin to approach the performance of multiple domain experts and surpass that of one expert. These findings demonstrate that an explanation system that has been given a well represented knowledge base can construct natural language responses whose quality approximates that of humans. More generally, they suggest that we are beginning to witness the appearance of computational machinery that will significantly broaden the bandwidth of human-computer communication.

Acknowledgments

We would like to thank our principle domain expert, Art Souther, for leading the knowledge base construction effort; Charles Callaway and the NLG students for their work on the realization system; Erik Eilerts, for building the knowledge base editing tools; Michael Elhadad, for generously assisting us with FUF; Peter Clark for his assistance with the statistical analysis; Dan Suthers, Rob Holte, and Paul Cohen for insights on the problems of evaluating explanation systems; the members of the Biology Knowledge Base Project: Liane Acker, Brad Blumenthal, Rich Mallory, Ken Murray, and Jeff Rickel; and Peter Clark, Charles Callaway, and the anonymous reviewers whose suggestions significantly improved the paper. Support for this research is provided by a grant from the National Science Foundation (IRI-9120310), a contract from the Air Force Office of Scientific Research (F49620-93-1-0239), and donations from the Digital Equipment Corporation.

References

Acker, Liane E. H. 1992. *Access Methods for Large, Multifunctional Knowledge Bases.* Ph.D. thesis, The University of Texas at Austin, Austin, Texas.

Acker, Liane E. H. and Bruce W. Porter. 1994. Extracting viewpoints from knowledge bases. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 547–552.

Appelt, Douglas E. 1985. Planning english referring expressions. *Artificial Intelligence*, 26:1–33.

Buchanan, Bruce G. and Edward H. Shortliffe, editors. 1984. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley, Reading, Massachusetts.

Callaway, Charles B. and James C. Lester. 1995. Robust natural language generation from large-scale knowledge bases. In *Proceedings of the Fourth Bar-Ilan Symposium on Foundations of Artificial Intelligence*, pages 96–105.

Carr, Brian and Ira P. Goldstein. 1977. Overlays: A theory of modelling for computer aided instruction. Technical Report AI Memo 406, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, February.

Cawsey, Alison. 1992. *Explanation and Interaction: The Computer Generation of Explanatory Dialogues.* MIT Press.

Cohen, Philip R. and C. Raymond Perrault. 1979. Elements of a plan-based theory of speech acts. *Cognitive Science*, 3:177–212.

Dale, Robert. 1992. *Generating Referring Expressions.* MIT Press.

Eilerts, Erik. 1994. KnEd: An interface for a frame-based knowledge representation system. Master's thesis, The University of Texas at Austin, Austin, Texas, May.

Elhadad, Michael. 1992. *Using Argumentation to Control Lexical Choice: A Functional Unification Implementation.* Ph.D. thesis, Columbia University.

Grimes, Joseph E. 1975. *The Thread of Discourse.* Mouton, The Hague.

Grosz, Barbara J. and Candace L. Sidner. 1986. Attention, intentions, and the structure of discourse. *Computational Linguistics*, 12(3):175–204.

Halliday, Michael A. K. and Ruqaiya Hassan. 1976. *Cohesion in English.* Longman, London.

Hobbs, Jerry R. 1985. On the coherence and structure of discourse. Technical Report CSLI-85-37, Center for the Study of Language and Information, Stanford University, Stanford, California, October.

Hovy, Eduard H. 1990. Pragmatics and natural language generation. *Artificial Intelligence*, 43:153–197.

Hovy, Eduard H. 1993. Automated discourse generation using discourse structure relations. *Artificial Intelligence*, 63:341–385.

Joshi, Aravind K. and Scott Weinstein. 1981. Control of inference: The role of some aspects of discourse structure—centering. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 385–387, Cambridge, England.

Kittredge, Richard, Tanya Korelsky, and Owen Rambow. 1991. On the need for domain communication knowledge. *Computational Intelligence*, 7(4):305–314.

Kukich, Karen. 1983. *Knowledge-Based Report Generation: A Knowledge Engineering Approach to Natural Language Report Generation.* Ph.D. thesis, University of Pittsburgh.

Lester, James. 1994. *Generating Natural Language Explanations from Large-Scale Knowledge Bases.* Ph.D. thesis, The University of Texas at Austin, Austin, Texas.

Lester, James C. and Bruce W. Porter. 1991a. A revision-based model of instructional multi-paragraph discourse production. In *Proceedings of the Thirteenth Cognitive Science Society Conference*, pages 796–800.

Lester, James C. and Bruce W. Porter. 1991b. A student-sensitive discourse generator for intelligent tutoring systems. In *Proceedings of the International Conference on the Learning Sciences*, pages 298–304, August.

Mann, William C. 1983. An overview of the Penman text generation system. In *Proceedings of the National Conference on Artificial Intelligence*, pages 261–265.

Mann, William C. and Sandra A. Thompson. 1987. Rhetorical structure theory: A theory of text organization. Technical Report ISI/RS-87-190, USC/Information Sciences Institute, Marina del Rey, California, June.

Maybury, Mark T. 1992. Communicative acts for explanation generation. *International Journal of Man-Machine Studies*, 37(2):135–172.

McCoy, Kathleen F. 1989 1990. Generating context-sensitive responses to object-related misconceptions. *Artificial Intelligence*, 41:157–195.

McKeown, Kathleen, Jaques Robin, and Karen Kukich. 1995. Generating concise natural language summaries. *Information Processing and Management.* Special Issue on Summarization.

McKeown, Kathleen R. 1985. *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text.* Cambridge University Press.

McKeown, Kathleen R., Myron Wish, and Kevin Matthews. 1985. Tailoring explanations for the user. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 794–798.

Mitchell, Kathleen P. 1992. A system to generate natural language sentences for explanations from a large knowledge base. Master's thesis, The University of Texas at Austin, Austin, Texas, May.

Mittal, Vibhu O. 1993. *Generating Natural Language Descriptions with Integrated Text and Examples.* Ph.D. thesis, University of Southern California, September.

Moore, Johanna D. 1995. *Participating in Explanatory Dialogues.* MIT Press.

Moore, Johanna D. and Cécile L. Paris. 1993. Planning text for advisory dialogues: Capturing intentional and rhetorical information. *Computational Linguistics*, 19(4):651–694.

Paris, Cécile L. 1988. Tailoring object descriptions to a user's level of expertise. *Computational Linguistics*, 14(3):64–78, September.

Porter, Bruce, James Lester, Kenneth Murray, Karen Pittman, Art Souther, Liane Acker, and Tom Jones. 1988. AI research in the context of a multifunctional knowledge base: The botany knowledge base project. Technical Report AI Laboratory AI88-88, University of Texas at Austin, Austin, Texas.

Rickel, Jeff and Bruce Porter. 1994. Automated modeling for answering prediction questions: Selecting the time scale and system boundary. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1191–1198.

Robin, Jaques. 1994. *Revision-Based Generation of Natural Language Summaries Providing Historical Background.* Ph.D. thesis, Columbia University, December.

Sibun, Penelope. 1992. Generating text without trees. *Computational Intelligence*, 8(1):102–122, March.

Souther, Art, Liane Acker, James Lester, and Bruce Porter. 1989. Using view types to generate explanations in intelligent tutoring systems. In *Proceedings of the Eleventh Cognitive Science Society Conference*, pages 123–130.

Stone, Brian A. and James C. Lester. 1996. Dynamically sequencing an animated pedagogical agent. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 424–431.

Suthers, Daniel D. 1988. Providing multiple views for explanation. In *Proceedings of the AAAI-88 Workshop on Explanation*, pages 12–15.

Suthers, Daniel D. 1991. A task-appropriate hybrid architecture for explanation. *Computational Intelligence*, 7(4):315–333, November.

Suthers, Daniel D. 1993. *An Analysis of Explanation and Its Implications for the Design of Explanation Planners.* Ph.D. thesis, University of Massachusetts, February.

Swartout, William R. 1983. XPLAIN: A system for creating and explaining expert consulting programs. *Artificial Intelligence*, 21:285–325.