Robust Natural Language Generation from Large-Scale Knowledge Bases[1]

Charles B. Callaway
(theorist@cs.utexas.edu)

Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712-1188

James C. Lester
(lester@adm.csc.ncsu.edu)

Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206

Content areas: Natural Language Generation, Large-Scale Knowledge Bases

Abstract

In recent years, the natural language generation community has begun to mature rapidly and produce sophisticated off-the-shelf surface realizers. A parallel development in the knowledge representation community has been the emergence of large-scale knowledge bases that house tens of thousands of facts encoded in expressive representational languages. Because of the richness of their representations and the sheer volume of their formally encoded knowledge, these knowledge bases offer the promise of significantly improving the quality of natural language generation. However, the representational complexity, scale, and task-independence of these knowledge bases pose great challenges to natural language generators.

We have designed, implemented, and empirically evaluated FARE, a *functional realization* system that exploits message specifications drawn from large-scale knowledge bases to create *functional descriptions*, which are expressions that encode both functional information (case assignment) and structural information (phrasal constituent embeddings). Given a message specification, FARE exploits lexical and grammatical annotations on knowledge base objects to construct functional descriptions, which are then converted to text by a surface generator. Two empirical studies—one with an explanation generator and one with a qualitative model builder—suggest that FARE is robust, efficient, expressive, and appropriate for a broad range of applications.

[1]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. This work was conducted at the University of Texas at Austin.

1 Introduction

In recent years, the field of natural language generation has begun to mature very rapidly. In addition to encouraging results in the form of specific theories and mechanisms that address particular generation phenomena, the field has witnessed the appearance of very sophisticated off-the-shelf surface realization systems [14, 4]. A parallel development in the knowledge representation community has been the emergence of large-scale knowledge bases (LSKBs) that house tens of thousands of facts encoded in expressive representational languages [20, 10]. Because of the richness of their representations and the sheer volume of their formally encoded knowledge, LSKBs offer the promise of significantly improving the quality of natural language generation. However, the representational complexity, scale, and task-independence of LSKBs pose great challenges to natural language generators.

The objective of our research is to develop techniques for expressive, robust natural language generation that can take advantage of these parallel developments in surface realization and large-scale knowledge base construction. To this end, we have designed, implemented, and empirically evaluated FARE,[2] a *functional realization* system that exploits message specifications drawn from large-scale knowledge bases to create *functional descriptions* [4, 5], which are expressions that encode both functional information (case assignment) and structural information (phrasal constituent embeddings). Given a message specification, FARE exploits lexical and grammatical annotations on knowledge base objects to construct functional descriptions, which are then converted to text by the FUF surface generator [4, 5].

We have conducted these investigations in the "laboratory" provided by the Biology Knowledge Base Project. The result of a seven year effort, the Biology Knowledge Base [20] is an immense, task-independent representation of more than 180,000 facts about botanical anatomy and physiology. (Its deductive closure is of course significantly larger.) To study both the robustness and range of applicability of our approach, FARE was evaluated in the context of two very different, knowledge-based systems, both of which extract structures from the Biology Knowledge Base: an explanation generator, KNIGHT [12, 13, 11], and a qualitative model constructor TRIPEL [21].

2 Functional Realization

Classically, natural language generation has been decomposed into two subtasks: *planning*, determining the content and organization of a text, and *realization*, translating the content to natural language. Although work has also been done on integrating the tasks so that decisions made at realization time can affect planning decisions [3, 1, 8], the two-task "pipeline" model has typically been adopted by work in multi-sentential natural language generation e.g., [15, 14, 16, 17, 18]. Realization itself can be

[2] Functional Assigner of Role Embeddings.

decomposed into two subtasks: *functional realization*, constructing functional descriptions from message specifications supplied by a planner; and *surface generation*, translating functional descriptions to text. Our work focuses on the design and implementation of functional realizers, which translate message specifications into functional descriptions that encode the appropriate 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.[3]

To illustrate, Figure 1 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 of the clause as well as for introducing additional phrasal substructures into the purpose clause. Our functional realizer takes full advantage of all of these features which are provided at the surface level by FUF and its accompanying grammar. Functional descriptions such as this one are produced by the functional realizer and passed to a surface generator for realization as text.

A functional realizer must create a mapping between the content units in a message specification and the constituents and semantic roles in functional descriptions. To properly realize the frames and relations in a message specification, a functional realizer must provide five central functionalities:

1. *Assign case roles to content units*: To achieve semantic equivalence, a functional realizer must ensure that the case roles in the message specification precisely match the case roles of the functional description being created. For example, the *ancestor-cell* slot in the message specification shown in Figure 2 should be mapped to the subject (functionally, the *agent*), and the *descendant-cells* slot should be mapped to the object of the infinitive (functionally, the *affected*).

2. *Organize content units into embedded phrase structures:* Content units may be complex; frequently they are modified by other relations and their values, but they are all bound together to represent component parts of a whole. The functional realizer guarantees that these components are properly packaged into a single unit. For example, the triple `(Megaspore-Mother-Cell`

[3]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-relater === ``during male gametophyte generation''))
```

Figure 1: A Functional Description

`contained-in Sporangium)` in Figure 2 must become an indivisible unit for the duration of this message specification. Indivisibility is important because without it, noun phrases which are modified by sub-frames could be changed. If the above triple were removed, then the *ancestor-cell* would incorrectly refer to any spore that originated from any megaspore mother cell, rather than to only spores that came from megaspore mother cells that were within sporangia.

3. *Provide local semantic information:* The functional realizer is responsible for checking for semantic conflicts between information retrieved from the lexicon and local semantic information which is specific to each type of message specification, and for resolving those conflicts. For example, when constructing a functional description for a message specification that should be realized as a *creative* sentence, as in Figure 2, a functional realizer must know that objects of an infinitival phrase should be indefinite, even though the default in the lexicon for the value of the *descendant-cells* slot is definite.

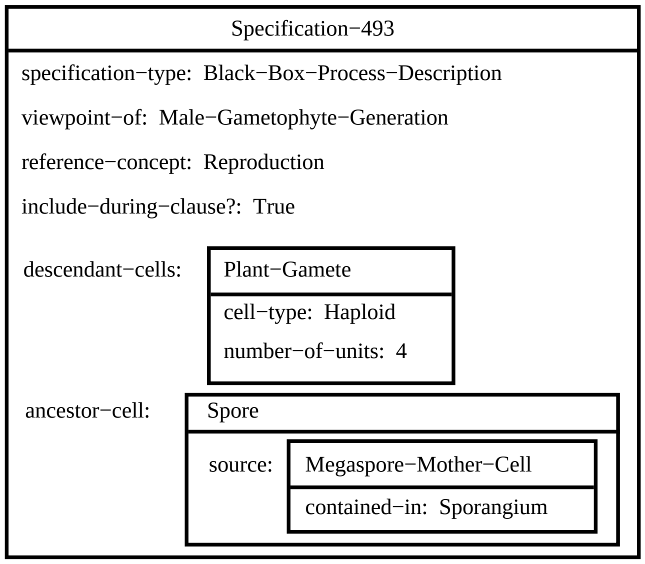

Figure 2: A Sample Message Specification

4. *Control the inclusion or exclusion of selected features:* Message specifications frequently contain meta-level information in the form of boolean variables which indicate whether or not to include supplementary sentence elements. In addition, a functional realizer must be able to compare elements of a sentence to eliminate potential redundancies. For example, the *Include-During-Clause?* relation specifies that the prefixed "During male gametophyte generation" phrase should be included as long as there are no lexical redundancies between it and the main verb "reproduce."

5. *Abort generation of sentences with defective message specifications*: When information on certain topics in a large-scale knowledge base is sparse, the message specifications will also be sparse. The functional realizer is responsible for ensuring that each message specification's required case roles exist. For example, if the *ancestor-cell* or *descendant-cells* slots in Figure 2 were empty, it would not be possible to convey the correct meaning of the specification without producing a sentence that appears either excessively vague or completely vacuous.

3 A Robust Functional Realization System

We have designed and implemented a functional realization system, FARE[4] (Figure 3), which provides the five key functionalities discussed above. FARE is given a message specification produced by a text planner. We have employed two text planners: an explanation generator, KNIGHT [12, 13, 11], and a qualitative model constructor TRIPEL [21] that has been "linguistically augmented." Both KNIGHT and TRIPEL employ the Biology Knowledge Base [20]. This is a large-scale knowledge base

[4] FARE's implementation consists of approximately 5,000 lines of Lucid Common Lisp.

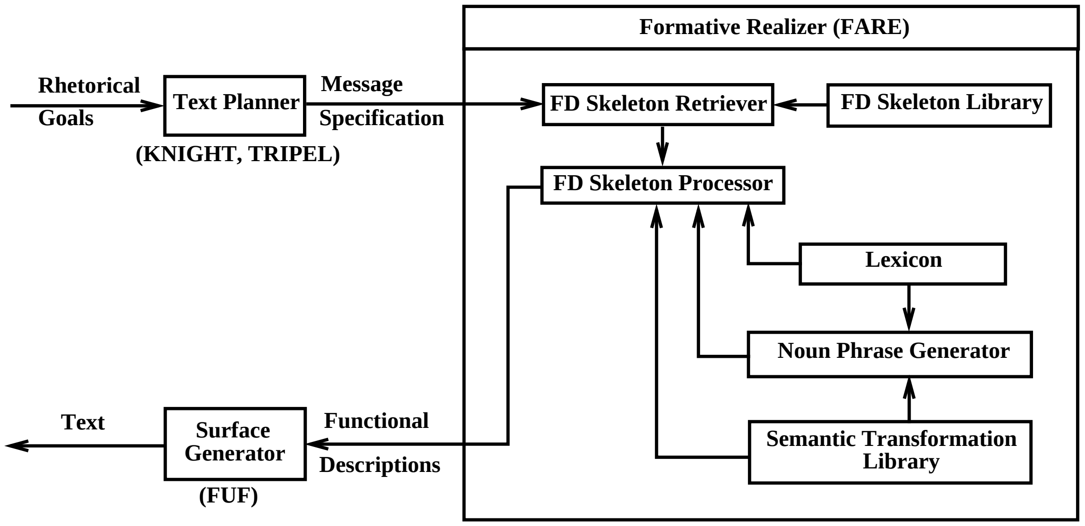

Figure 3: An Architecture for Functional Realization

whose hierarchy encodes information about biological objects (representing botanical anatomy) and biological processes (representing botanical physiology and development). This knowledge base is an immense structure that contains more than 180,000 facts and is one of the largest in existence.

Given a message specification, FARE uses its knowledge of case mappings, syntax, and lexical information to construct a functional description. FARE then passes the functional description to FUF, a unification-based surface generator [4, 5]. FUF is accompanied by an extensive, portable English grammar, SURGE,[5] which Elhadad describes as "the result of five years of intensive experimentation in grammar writing." FUF's grammar borrows the notions of feature structures and unification from *functional unification grammars* [9]. It also incorporates the notion of systems from *systemic grammars* [7]. FUF frees a functional realizer from lower-level issues such as positioning and morphological manipulation. After discussing FARE's principal knowledge sources, we describe the algorithms that guide its operation.

3.1 Knowledge Sources

FARE employs three principle knowledge sources: a lexicon, a library of FD-Skeletons, and a library of semantic transformations. We discuss each of these in turn. To provide immediate access to lexical information, FARE employs a distributed lexicon: each concept contains all of the lexical information the concept requires for realization. Each concept in the lexicon may include several lexical features. All lexical entries include the *lexical type* of the concept, which provides the key to interpreting all of

[5] SURGE is the largest "generation" grammar in existence.

```
((cat  clause)
 (proc  ((type  <VERB-TYPE>)
         (verb  <VERB-STRING>)))
 (partic  ((agent  <TRANSPORTER-FD>)
           (affected  <TRANSPORTED-ENTITIES-FD>)
           (location  ((cat  list)
                       (distinct  (<SOURCE-FD>
                                   <CONDUIT-FD>
                                   <DESTINATION-FD>))))))
 (time-relater  ''during <PROCESS-LEXEME>''))
```

Figure 4: The Template of an FD-Skeleton for *Transportation*

the lexical information present on the concept. It can contain either a specific grammatical constituent, e.g., NPs, VPs, PPs, AdjPs, etc., or a subclass of constituents, such as relative clauses and possessive phrases. Many different types of concepts use *lex-FD* feature, which contains a functional description that expresses the given concept.

Concepts that are nouns may have a *noun* feature, which indicates the concept's lexeme. "Single word" noun concepts, e.g., "embryo" or "stem," have a default case, and are thus optimized and separated from the more complex nouns that can be described by *lex-FD*. The default case can be modified to alter features such as definiteness and countability. The *mass-or-count?* feature stores such countability information to override the default value *count*. Adjectives and adverbs have their own features as well.[6] To create exceptions in the lexical ontology, the lexical access methods exploit inheritance mechanisms to find the most specific verb available, e.g, "elongate" instead of "grow."

The second knowledge source used by the functional realizer is a library of FD-Skeletons. An FD-Skeleton is a template that encodes the deep structure represented by a functional description. Not only do the values that fill the variables in FD-Skeletons have to pass semantic tests, but sometimes entire feature sets of the functional description are conditional as well. Some FD-Skeletons encode the case structure for processes. For example, we have developed FD-Skeletons for 20 processes that play a central role in biology, e.g., assimilation, development, reproduction, and transportation. Figure 4 shows the template of the FD-Skeleton for producing functional descriptions of transportation processes.[7] This FD-Skeleton can be used to construct a sentence such as, "During pollen grain transfer, a pollen grain is transferred from the anther to the stigma."

[6]Adverbs are typically encountered as slot annotations in message specifications.

[7]Note that the template is merely one of two components of FD-Skeletons; Skeletons also include semantic tests for inclusion and modification of nested functional descriptions.

Following in the NLG "revision" tradition [24, 15, 22, 23, 6, 12], the third knowledge source used by the functional realizer is a library of semantic transformations. For example, when given the triples `(Water amount 4)` and `(H+ amount 3)`, without access to semantic transformations, the system would return the functional descriptions representing the strings "4 portions of water" and "4 portions of hydrogen ion." The job of the semantic transformations is first to detect this sort of problem and second to correct it by transforming functional descriptions. In this example, the system must determine that the *amount* relation functions differently for mass and count nouns, and appropriately return "4 hydrogen ions" in the latter case.

3.2 FD-Skeleton Retrieval and Processing

The FD-Skeleton Retriever obtains the appropriate FD-Skeleton by indexing into its library. If the topic of the specification is a process, the FD-Skeleton Retriever exploits the taxonomy of the knowledge base to locate the most specific case structure that can be used. Otherwise, the retrieval process requires no inference because each specification type points to a unique FD-Skeleton. For example, in Figure 2 the values of the *specification-type* and *reference-concept* relations provide the needed indices into the FD-Skeleton Library.

Next, the FD-Skeleton Processor determines if each of the essential slots are present; if any of these tests fail, it will note the deficiency and abort. If the message is well-formed, the FD-Skeleton Processor uses the FD-Skeleton as a template for forming a functional description. It instantiates the variables in the FD-Skeleton, each of which is associated with a particular attribute name that appears in the message specification. For each variable in the FD-Skeleton, the FD-Skeleton Processor obtains the name of an attribute (or a group of similar attributes) on the message specification. For example, the FD-Skeleton that is used to realize a "structural" message specification exploits the relation hierarchy in the knowledge base to obtain all "part" attributes in the message specification, e.g., *parts*, *composed of*, *contains*, etc.

The FD-Skeleton Processor retrieves the values that appear on the selected attributes of the message specification. These values are then used to instantiate variables in the FD-Skeleton. For example, in Figure 2 the *descendant-cells* and *ancestor-cell* relations are chosen, and their values are computed by the Noun Phrase Generator and inserted into the template. In addition, some relations that appear in a message specification, e.g., *include-during-clause?*, are directives to the FD-Skeleton Processor instead of attributes. These directives indicate to the FD-Skeleton Processor that it should include (or exclude) particular subordinate clauses that the text planner has deemed necessary (or burdensome).

Next, the FD-Skeleton Processor invokes the Noun Phrase Generator (described below) to construct a functional description representing the noun phrase that expresses those values. The FD-

Skeleton Processor then binds the resulting noun phrase functional descriptions to the variable and pushes this pair onto the binding list. Finally, it splices all of the new noun phrases it has constructed into the FD-Skeleton by substituting the appropriate noun-phrase functional descriptions for each variable. As it instantiates the variables in the evolving functional description, it checks for expressions that would produce semantically correct but "lexically redundant" sentences. This check prevents the generation of sentences such as, "During cell formation, a cell forms."

3.3 Noun Phrase Generation

The Noun Phrase Generator (Figure 5) is given a list of concepts, which are the values that appear on the attributes of a message specification. It is also given a context, which initially is null. If the Noun Phrase Generator is given more than one concept to translate into a noun phrase, it recursively invokes itself on each of the concepts. This will produce a conjoined list of functional descriptions, each of which represents a noun phrase for one of the concepts.

Next, the Noun Phrase Generator obtains the type of functional descriptions that comprise the group. It then constructs an "enclosing" functional description based on the type, and it embeds each of the functional descriptions resulting from the recursive invocations in the "enclosing" functional description and finally returns this entire expression. If it is invoked with a single concept—this is the "base" case—it first obtains the lexical information associated with the concept. For example, in Figure 2, the simple concepts Plant-Gamete and Spore are such base cases.

Its next task is to determine how this general lexical information should be modified by additional information known about the concept, i.e., it must augment the basic lexical information with lexical information about the concept's attributes. To do so, it obtains four types of attributes that may appear on the concept: *describer* attributes, e.g., color; *cardinal* attributes, e.g., number-of-units; *relative clause* attributes, e.g., covered-by; and *partitive* attributes, e.g., subregions. A given concept may be modified by all or none of these types of attributes. For example, in Figure 2, Plant-Gamete and Spore are modified in this manner by the attributes which appear below them. Cell-type is a *describer* attribute for Plant-Gamete while amount is a *cardinal* attribute. Additionally, since the subframe beginning at Megaspore-Mother-Cell is itself a complete subframe, the Noun Phrase Generator calls itself recursively and uses the entire resulting functional description as the value for the *source* relation above it.

For each type that the Noun Phrase Generator encounters, it constructs a functional description of that group. Each of these specialized construction functions may recursively invoke the algorithm and augment the existing context with information about the current phrase type. For example, the Noun Phrase Generator may augment a recursive call with the "number" of the enclosing noun phrase. In this case, the augmentation permits the system to propagate number information to substructures

```
MAKE-NOUN-PHRASE ($Concept\text{-}List$, $Context$)

if length ($Concept\text{-}List$) > 1 then
  $Functional\text{-}Description\text{-}List \leftarrow \emptyset$
  for each $Concept$ in $Concept\text{-}List$ do
      $New\text{-}Noun\text{-}Phrase \leftarrow$ make-noun-phrase (($Concept$))
      $Functional\text{-}Description\text{-}List$
                  $\leftarrow$ enqueue ($New\text{-}Noun\text{-}Phrase$,
                                 $Functional\text{-}Description\text{-}List$)
  $FD\text{-}Group\text{-}Type \leftarrow$ get-FD-group-type ($Functional\text{-}Descriptions$)
  $Result\text{-}FD \leftarrow$ make-complex-noun-phrase ($Functional\text{-}Description\text{-}List$,
                                              $FD\text{-}Group\text{-}Type$)
  return ($Result\text{-}FD$)
else
    $Concept \leftarrow$ first ($Concept\text{-}List$)
    $Functional\text{-}Description \leftarrow$ compute-lex-information ($Concept$, $Context$)
    $Concept\text{-}Attributes \leftarrow$ get-concept-attributes ($Concept$)
    $Describer\text{-}Attributes \leftarrow$ get-describer-attributes ($Concept\text{-}Attributes$)
    $Cardinal\text{-}Attributes \leftarrow$ get-cardinal-attributes ($Concept\text{-}Attributes$)
    $Rel\text{-}Clause\text{-}Attributes \leftarrow$ get-rel-clause-attributes ($Concept\text{-}Attributes$)
    $Partitive\text{-}Attributes \leftarrow$ get-partitive-attributes ($Concept\text{-}Attributes$)
    $Describer\text{-}FD \leftarrow$ make-describer-FD ($Concept$, $Describer\text{-}Attributes$,
                                          $Context$)
    $Cardinal\text{-}FD \leftarrow$ make-cardinal-FD ($Concept$, $Cardinal\text{-}Attributes$,
                                        $Context$)
    $Relative\text{-}Clause\text{-}FD \leftarrow$ make-rel-clause-FD ($Concept$,
                                                  $Rel\text{-}Clause\text{-}Attributes$,
                                                  $Context$)
    $Partitive\text{-}FD \leftarrow$ make-partitive-FD ($Concept$, $Partitive\text{-}Attributes$,
                                          $Context$)
    $Result\text{-}FD \leftarrow$ merge-FDs ($Functional\text{-}Description$,
                               $Describer\text{-}FD$, $Cardinal\text{-}FD$,
                               $Relative\text{-}Clause\text{-}FD$, $Partitive\text{-}FD$)
    return ($Result\text{-}FD$)
```

Figure 5: The MAKE-NOUN-PHRASE Algorithm

such as relative clauses, whose verb endings are influenced by the number feature of the enclosing noun phrase. Finally, the Noun Phrase Generator merges the resulting functional descriptions into the original lexical information (also a functional description) and returns this expression to the FD-Skeleton Processor. This functional description expresses as a noun phrase the concept(s) that the FD-Skeleton Processor encountered in its traversal of the message specification.

In the absence of semantic transformations, the result of this entire process in our example from Figure 2 would be a functional description that would produce the sentence, "During male gametophyte generation, the spore, which originates in the megaspore mother cell, which is contained in the sporangium, reproduces to form 4 plant gametes." However, the system finds two applicable transformations, which are triggered when the system encounters two adjacent relative clauses: the first transformation collapses the relative clause "which originates in" to the preposition "from"; the second transformation collapses the relative clause "which is contained in" to the preposition "in." The final resulting functional description is shown in Figure 1. FARE passes this to FUF, which realizes it as, "During male gametophyte generation, the spore from the megaspore mother cell in the sporangium reproduces to form four haploid plant gametes."

4 Evaluation

Because the conclusions of empirical studies should be considerably less equivocal than those derived from "proof-of-concept" systems, we have taken an empirical approach to evaluation. With three significant exceptions ([8], [2], and [19]), the field of natural language generation has not witnessed the development of an "empiricist evaluation" school. However, it is clear that empirical evaluations can yield very informative data. To this end, we conducted a formal evaluation of FARE in conjunction with an explanation generator and an informal evaluation in conjunction with a qualitative model builder.

4.1 Robustness, Efficiency, and Expressiveness

First, we evaluated FARE with KNIGHT [12, 13, 11], a robust explanation generator that constructs explanations about scientific phenomena. It extracts structures from a large-scale knowledge base and organizes them into hierarchical discourse plans. Working in conjunction, KNIGHT, FARE, and FUF have produced more than a thousand different sentences. On average FARE requires approximately 1–2 seconds on a DEC Alpha to produce a functional description.

Our formal study employed two panels of domain experts. Experts on the first panel served as "writers," i.e., they produced explanations in response to questions. Experts on the second panel served as "judges," i.e., they analyzed different dimensions of explanations and assigned grades. It

Generator	*Overall*	*Content*	*Organization*	*Writing*	*Correctness*
System	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 1: Comprehensive Analysis

is important to note that *none of the judges were informed about the purpose of the study, and none were aware that they were judging computer-generated explanations.* Judges were asked to rate the explanations on several dimensions, including overall quality and writing style. To provide judges with a familiar rating scale, they were asked to assign letters grades (A, B, C, D, or F) to each dimension of the explanation. We assigned explanations to judges using an allocation policy that obeyed the following four constraints: each judge received explanations that were approximately evenly divided between objects and processes; each judge received explanations that were approximately evenly divided between those that were produced by Knight and those that were produced by biologists; no judge received two explanations of the same concept; and the explanations written by each writer were not evaluated by at least two judges.

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 subscores of content, organization, writing style, and correctness. On the overall rating and on each of the subscores, the system scored within approximately "half a grade" of the biologists (Table 1). (In the tables, ± denotes the standard error, i.e., the standard deviation of the mean.) Given these results, we decided to investigate the differences between the Knight-Fare grades and the biologists' grades. When we normalized the grades by defining an "A" to be the mean of the biologists' grades, the system earned approximately a B–B^+. The results demonstrate that the quality of Fare's text has begun to approach that of humans.

4.2 Range of Applicability

A second study provides evidence that Fare has a broad range of applicability. To investigate Fare's ability to realize message specifications that were generated for a significantly different kind of application, we obtained a qualitative model constructor, Tripel [21]. In collaboration with the designer of Tripel, we modified it to produce text planning commands in addition to its simulation data.

Next, we extended FARE's library of Functional Description Skeletons to address the new specification types. FARE used message specifications produced by the augmented qualitative model constructor to generate functional descriptions for sentences such as, "The rate of ABA synthesis in the plant's mesophyll cells is influenced by one thing: it is negatively affected by the turgor pressure in the plant's mesophyll cells." Within three weeks, we were able to extend FARE to produce completely correct functional descriptions for this new application: the text it generated was favorably evaluated by both the domain expert and the designer of the qualitative model building system.

5 Conclusion and Future Work

We have designed, implemented, and evaluated FARE, a robust functional realization system. By exploiting an expressive lexicon that is distributed throughout the knowledge base, as well as libraries of functional description skeletons and semantic transformations, FARE uses message specifications drawn from a large-scale knowledge base to create functional descriptions. The functional descriptions are then realized in text by FUF, a sophisticated surface generator. FARE provides five key functionalities: it assigns case roles to content units, organizes content units into embedded phrase structures, provides local semantic information, controls the inclusion (or exclusion) of selected features, and aborts generation when it encounters defective message specifications. Two empirical studies suggest that FARE is robust, efficient, capable of producing quality text, and appropriate for a broad range of applications.

Encouraged by the experimental results we have obtained, we are continuing to enhance FARE's capabilities. On a theoretical front, we are considering whether it may be possible to cast functional realization as a unification problem.[8] If we are successful in this venture, then FARE can be more cleanly reimplemented within a second FUF image. On a practical front, it has become clear that we must semi-automate the creation and maintenance of FARE's realization libraries and lexical information. Work is currently under way to create integrated tools that will allow knowledge engineers with minimal linguistic expertise to create lexicon entries and updates; another tool is planned that will automatically update the components of FARE itself. Both of these tools will be implemented with a GUI and will guide the knowledge engineer by continuously displaying acceptable linguistic choices and incrementally realizing the accumulating structure at each decision point.

Acknowledgements

We would like to thank: Bruce Porter for leading the Biology Knowledge Base project and for providing comments on earlier drafts of this paper; Kathy Mitchell, Rich Jones, and Teresa Chatkoff for their

[8] Thanks to Michael Elhadad for suggesting this approach.

work on FARE; Michael Elhadad, for developing and generously assisting us with FUF; Art Souther, our principle domain expert; Erik Eilerts, for building the knowledge base editing tools; Peter Clark, for assistance with the evaluation; Jeff Rickel, the designer of TRIPEL; and the other members of the Biology Knowledge Base Project, Liane Acker, Brad Blumenthal, Rich Mallory, and Ken Murray.

References

[1] D. Appelt. Planning english referring expressions. *Artificial Intelligence*, 26:1–33, 1985.

[2] A. Cawsey. *Explanation and Interaction: The Computer Generation of Explanatory Dialogues.* MIT Press, 1992.

[3] L. Danlos. Conceptual and linguistic decisions in generation. In *Tenth International Conference on Computational Linguistics*, pages 501–504, Stanford University, July 1984.

[4] M. Elhadad. FUF: The universal unifier user manual version 5.0. Technical Report CUCS-038-91, Department of Computer Science, Columbia University, 1991.

[5] M. Elhadad. *Using Argumentation to Control Lexical Choice: A Functional Unification Implementation.* PhD thesis, Columbia University, 1992.

[6] R. P. Gabriel. Deliberate writing. In D. D. McDonald and L. Bolc, editors, *Natural Language Generation Systems*, pages 1–46. Springer-Verlag, New York, 1988.

[7] M. Halliday. *System and Function in Language.* Oxford University Press, Oxford, 1976.

[8] E. Hovy. Pragmatics and natural language generation. *Artificial Intelligence*, 43:153–197, 1990.

[9] M. Kay. Functional grammar. In *Proceedings of the Berkeley Linguistic Society*, 1979.

[10] D. Lenat and R. Guha. *Building Large Knowledge Based Systems.* Addison-Wesley, Reading, Massachusetts, 1990.

[11] J. Lester. *Generating Natural Language Explanations from Large-Scale Knowledge Bases.* PhD thesis, The University of Texas at Austin, Austin, Texas, 1994.

[12] J. Lester and B. Porter. A revision-based model of instructional multi-paragraph discourse production. In *Proceedings of the Thirteenth Cognitive Science Society Conference*, pages 796–800, 1991.

[13] J. Lester and B. Porter. A student-sensitive discourse generator for intelligent tutoring systems. In *Proceedings of the International Conference on the Learning Sciences*, pages 298–304, August 1991.

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

[15] W. Mann and J. Moore. Computer generation of multiparagraph english text. *American Journal of Computational Linguistics*, 7(1):17–29, 1981.

[16] D. McDonald. Description directed control: Its implications for natural language generation. *Computers and Mathematics*, 9(1):111–130, 1983.

[17] K. McKeown. *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text.* Cambridge University Press, 1985.

[18] M. Meteer. *The Generation Gap: The Problem of Expressibility in Text Planning.* PhD thesis, University of Massachusetts, February 1990.

[19] V. Mittal. *Generating Natural Language Descriptions with Integrated Text and Examples.* PhD thesis, University of Southern California, September 1993.

[20] B. Porter, J. Lester, K. Murray, K. Pittman, A. Souther, L. Acker, and T. Jones. 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, 1988.

[21] J. Rickel and B. Porter. 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, 1994.

[22] M. M. Vaughan and D. D. McDonald. A model of revision in natural language generation. In *Proceedings of the 24th Annual Meeting*, pages 90–96, Columbia University, 1986. Association for Computational Linguistics.

[23] W.-K. C. Wong and R. F. Simmons. A blackboard model of text production with revision. In *Proceedings of the AAAI Workshop on Text Planning and Realization*, St. Paul, Minnesota, August 1988.

[24] M. Yazdani. Reviewing as a component of the text generation process. In G. Kempen, editor, *Natural Language Generation*, pages 183–190. Martinus Nijhoff, Dordrecht, The Netherlands, 1987.