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

Abstract

We have begun to see the emergence of large-scale knowledge bases that house tens of thousands of facts encoded in expressive representational languages. The richness of these representations offer the promise of significantly improving the quality of natural language generation, but their representational complexity, scale, and task-independence pose great challenges to 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 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 sophisticated, generic surface realization systems [12, 2]. 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 [14, 8]. 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

[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 research was conducted at the University of Texas at Austin.

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 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* [2, 3], which are expressions that encode both functional information (case assignments) and structural information (phrasal constituent embeddings). Given a message specification, FARE constructs functional descriptions, which are then converted to text by the FUF surface generator [2, 3]. 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 [14] is an immense, task-independent representation of more than 180,000 facts about botanical anatomy and physiology. To study FARE's robustness and range of applicability, it was evaluated with two very different text planners: an explanation generator, KNIGHT [10, 11, 9], and a qualitative model constructor TRIPEL [15], which was augmented to produce text plans.

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. 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. Our work focuses on the design and implementation of functional realizers, which translate message specifications into functional descriptions that encode case assignments and phrasal constituent embeddings.

Figure 1 depicts a sample functional description (FD). The first line, `(cat clause)`, indicates that what follows will be some type of verbal phrase. The second line contains the keyword `proc`, which denotes that everything in its scope will describe the structure of the 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`, which is further defined by its lexical entry and a modifying prepositional phrase indicated by the keyword `qualifier`. The structure beginning with `circum` describes a subordinate infinitival purpose clause.

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

[2] Functional Assigner of Role Embeddings.

```
((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

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 relations in the message specification are precisely mapped to case roles in an FD, e.g., *ancestor-cell* in Figure 2 should be mapped to *agent*.

2. *Organize content units into embedded phrase structures:* A functional realizer must guarantee that complex content units are are properly packaged in FDs. For example, `(Megaspore-Mother-Cell contained-in Sporangium)` in Figure 2 should become an indivisible syntactic unit.

3. *Provide local semantic information:* A functional realizer is responsible for detecting and resolving semantic conflicts between lexical information and local semantic information on a message specification, e.g., when constructing an FD for a *creative* sentence, such as Figure 2 calls for, it must know that infinitival objects should be indefinite rather than the default value of definite.

4. *Control the inclusion of selected features:* Message specifications frequently contain meta-level information, e.g., *Include-During-Clause?* specifies that a time-relater should be included if there are no lexical redundancies between it and the head verb.

5. *Abort generation of sentences with defective message specifications*: A functional realizer is responsible for detecting missing case roles required for proper realization.

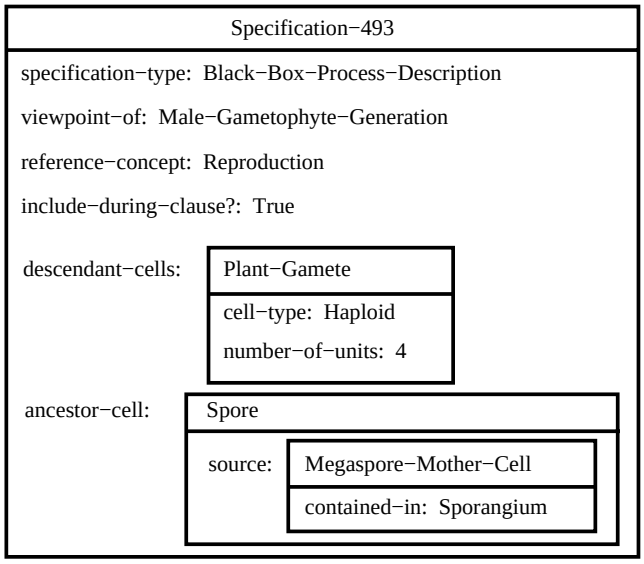

Figure 2: A Sample Message Specification

3 A Robust Functional Realization System

We have designed and implemented a functional realization system, FARE[3] (Figure 3), which provides the five key functionalities discussed above. Given a message specification produced by a text planner, FARE uses its knowledge of case mappings, syntax, and lexical information to construct an FD, which it passes to FUF, a unification-based surface generator [2, 3].[4] FARE was developed with text planners that employ the Biology Knowledge Base [14], which contains more than 180,000 facts about botanical anatomy and physiology. We have employed two text planners: an explanation generator, KNIGHT [10, 11, 9], and a qualitative model constructor TRIPEL [15] that has been "linguistically augmented."

3.1 Knowledge Sources

FARE employs three principle knowledge sources: a lexicon, a library of FD-Skeletons, and a library of semantic transformations. Each concept in the lexicon may include several lexical features, and all lexical entries include the *lexical type* of the concept, which indicates either a specific grammatical constituent, e.g., NP, or a subclass of constituents, such as relative clauses. In addition to lexemes and type information, the lexicon provides features such as *mass-or-count?* to store countability information that overrides default values. 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."

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

[4] FUF is accompanied by an extensive, portable English grammar, SURGE (the largest "generation" grammar in existence), described by Elhadad as "the result of five years of intensive experimentation in grammar writing." SURGE borrows the notions of feature structures and unification from *functional unification grammars* [7] and systems from *systemic grammars* [5].

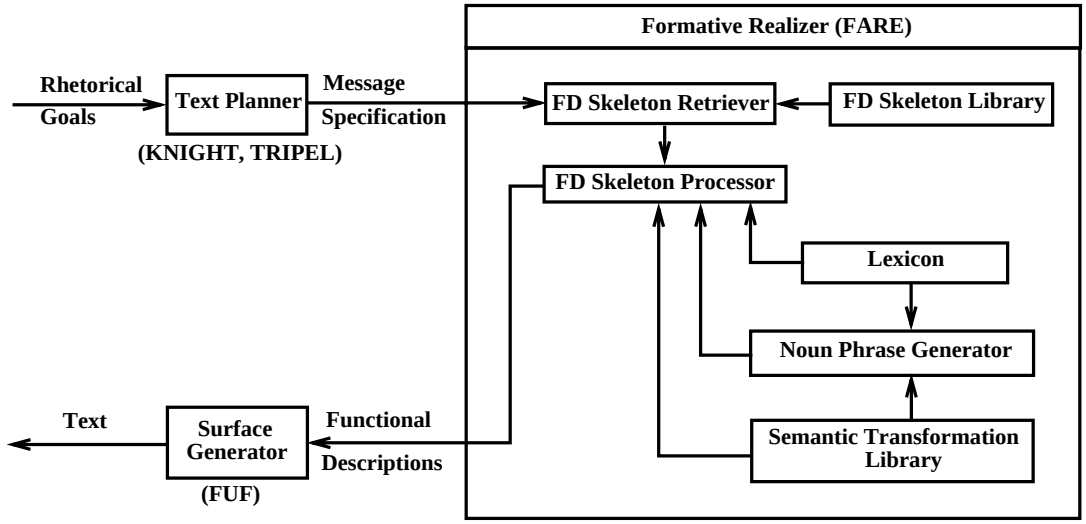

Figure 3: An Architecture for Functional Realization

The second knowledge source used by the functional realizer is a library of FD-Skeletons. An FD-Skeleton consists of (1) a collection of semantic tests for inclusion and modification of nested FDs, and (2) a template that encodes the deep structure represented by an FD. Among the more important of various types of FD-Skeletons are those for describing processes. Currently we have developed FD-Skeletons for over 20 processes that play a central role in biology, e.g., assimilation, development, and reproduction. Figure 4 shows the template of the FD-Skeleton for producing functional descriptions of transportation processes.[5]

Following in the NLG "revision" tradition [16, 17, 4, 10], 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 FDs representing the strings "4 portions of water" and "3 portions of hydrogen ion." Semantic transformations detect and correct problems of this sort. In this example, they determine that *amount* behaves differently for mass and count nouns, and they appropriately return "3 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

[5]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.

```
((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*

no inference because each specification type points to a unique FD-Skeleton. 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 an FD. It instantiates the variables in the FD-Skeleton, each of which is associated with a particular attribute 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 obtains all "part" attributes in the message specification, e.g., *parts* and *composed-of*.

The FD-Skeleton Processor then retrieves the values that appear on the selected attributes of the message specification. These values are 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. Next, the Noun Phrase Generator (described below) is invoked to construct an FD representing the noun phrase expressing those values. Finally, the FD-Skeleton Processor splices all of the new noun-phrase FDs into the FD-Skeleton.

3.3 Noun Phrase Generation

The Noun Phrase Generator (Figure 5) is given a list of concepts, which are the values of the attributes of a message specification. If the Noun Phrase Generator is given more than one concept, it recursively invokes itself on each of the concepts, thereby producing a conjoined list of FDs, each of which represents a noun phrase for one of the concepts. Next, it obtains the type of FDs that comprise the group. It then constructs an "enclosing" FD based on the type, and it embeds each of the FDs resulting from the recursive invocations in the "enclosing" FD and finally returns this entire expression. If it is invoked

```
MAKE-NOUN-PHRASE (Concept-List, Context)

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

Figure 5: The MAKE-NOUN-PHRASE Algorithm

with a single concept, it first obtains the lexical information associated with the concept.

Its next task is to 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. For example, in Figure 2, cell-type is a *describer* attribute for Plant-Gamete while amount is a *cardinal* attribute. For each type that the Noun Phrase Generator encounters, it constructs an FD 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 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 FDs into the original lexical information (also an FD) and returns this expression to the FD-Skeleton Processor.

The final resulting FD for the message specification depicted in Figure 2 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.[6] 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. First, we evaluated FARE with KNIGHT [10, 11, 9], a robust explanation generator that constructs explanations about scientific phenomena. Working in conjunction, KNIGHT, FARE, and FUF have produced more than a thousand different sentences,[7] including the following: "During egg fertilization, an angiosperm sperm cell fertilizes a plant egg cell to form a zygote."; "Embryo sac development is a step of embryo sac formation, which is a step of reproduction."; "Sporogenesis occurs immediately before gametophyte generation."; "The root system is part of the plant and is connected to the mainstem."; "The subregions of the root system include the meristem, which is where root system growth occurs."; and, "During sperm cell transport, 2 angiosperm sperm cells are transported from the pollen tube to the embryo sac."

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. *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. Using an A–F (4–0) scale, the system scored within approximately "half a grade" of the biologists (Table 1).[8]

A second study provides evidence that FARE has a broad range of applicability. To investigate FARE's ability to realize message specifications for a significantly different kind of task, we augmented a qualitative model constructor, TRIPEL [15], with text planning commands. FARE used message specifications produced by the augmented qualitative model constructor to generate FDs 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 FDs for this new application: the text it generated

[6] With three significant exceptions ([6], [1], and [13]), the field of natural language generation has not witnessed the development of an "empiricist evaluation" school.

[7] On average, FARE requires 1–2 seconds on a DEC Alpha to produce an FD.

[8] In the tables, ± denotes the standard error, i.e., the standard deviation of the mean.

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

was favorably evaluated by both the domain expert and the designer of the qualitative modeling system.

5 Conclusion

We have designed, implemented, and evaluated FARE, a robust functional realization system. By exploiting an expressive lexicon, 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, which are then realized in text by a sophisticated surface generator, FUF. 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 of selected features, and detects defective message specifications. Two empirical studies suggest that FARE is robust, efficient, capable of producing quality text, and appropriate for a broad range of text planners.[9]

References

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

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

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

[4] 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.

[9] We would like to thank: Bruce Porter for leading the Biology Knowledge Base project and for providing comments on earlier drafts of this paper; Michael Elhadad, for developing and generously assisting us with FUF; Kathy Mitchell, Rich Jones, and Teresa Chatkoff for their work on FARE; 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.

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

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

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

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

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

[10] 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.

[11] 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.

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

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.