Software Re-Use and Evolution in Text Generation Applications[‡]

Karen Kukich[*] and **Rebecca Passonneau**[*†] and **Kathleen McKeown**[†] and **Dragomir Radev**[†] and **Vasileios Hatzivassiloglou**[†] and **Hongyan Jing**[†]

[*]Bellcore
445 South Street
Morristown, NJ 07960, USA
{kukich, beck}@bellcore.com

[†]Department of Computer Science
450 Computer Science Building
Columbia University
New York, NY 10027, USA
{becky, kathy, radev, vh, hjing}@cs.columbia.edu

Abstract

A practical goal for natural language text generation research is to converge on a separation of functions into modules that can be independently re-used. This paper addresses issues related to software re-use and evolution in text generation systems. We describe the benefits we obtained by adapting and generalizing the generation modules and techniques we used for the successive development of three distinct text generation applications, PLANDoc, FlowDoc, and ZEDDoc. We suggest that design principles such as the use of a common, modular pipeline architecture, a consistent and general data representation format, and domain-independent algorithms for generation subtasks, together with component re-use and adaptation, facilitate both application development and research in the field. In our experience, these principles led to significant reductions in development time for successive applications, from three years to one year to six months, respectively. They also enabled us to isolate domain-specific knowledge and devise reusable, domain-independent algorithms for generation tasks such as ontological generalization and discourse structuring.

[‡]The authors wish to acknowledge Jacques Robin, James Shaw, Jong Lim, and Larry Lefkowitz, who also played essential roles in the design and development of PLANDoc and FlowDoc.

1 Introduction

Recent technological advances, such as the widespread use of the World Wide Web and ready access to a multitude of extensive large-scale databases, have created novel opportunities for practical text generation applications. At the same time, to take full advantage of these opportunities, text generation systems must be easily adaptable to new domains, changing data formats, and distinct underlying ontologies.

One crucial factor contributing to the generalization and subsequent practical and commercial viability of text generation systems is the adaptation and re-use of text generation modules and the development of re-usable tools and techniques. In this paper, we focus on the lessons learned during the successive development of three text generation systems at Bellcore: PLANDoc (McKeown et al., 1994) summarizes execution traces of an expert system for telephone network capacity expansion analysis; FlowDoc (Passonneau et al., 1996) provides summaries of the most important events in flow diagrams constructed during business reengineering; and ZEDDoc (Passonneau et al., 1997) produces summaries of activity for a user-specified set of advertisements within a user-specified time period from logs of WWW page hits.

We built FlowDoc and ZEDDoc by adapting components of the PLANDoc system. The transfer of the original PLANDoc modules to new domains led to the replacement of some hard-coded rules and ontological knowledge with more general, domain-independent components. This encapsulation, or "plug-and-play" feature, enabled the transfer of many of FlowDoc's modules to ZEDDoc with minimal alterations. As a result, development time was significantly reduced — from three years for PLANDoc to one year for FlowDoc to six months for ZEDDoc.

In the remainder of the paper, we provide background information on the three systems and then present and discuss four design principles that facilitate the development of text generation systems and their portability to new domains and applications:

- A common, stable pipeline architecture that subdivides generation tasks (e.g., sentence planning or lexical choice) into separate modules.
- A consistent and general data representation that allows easy interfacing between generation modules and between the text generator and external sources (e.g., relational databases).
- Domain-independent methods for performing each generation subtask, that avoid hard-coded knowledge and rely instead on external, plug-and-play knowledge bases.
- Component re-use and adaptability from each application to the next, with the aim of improving generality and achieving the data independence goal described previously.

2 Background

PLANDoc (McKeown et al., 1994), the first major text generation system developed at Bellcore, is an enhancement to Bellcore's LEIS-PLAN™ network planning product. Human engineers use LEIS-PLAN to do network capacity expansion studies, during which they explore alternative scenarios to arrive at an optimal configuration of equipment that meets demands for new services while minimizing costs. PLANDoc produces textual summaries of the scenarios explored by engineers. It transforms lengthy execution traces into human-readable summaries by making heavy use of conjunction, ellipsis, and paraphrasing. It also allows engineers to intersperse their own comments and justifications while using the tool. PLANDoc is currently in widespread use throughout the Southwestern Bell Corporation and has been requested by at least two other regional Bell companies. As an example, Figure 1 shows a fragment of the input to PLANDoc for a particular study, PLANDoc's representation of the same information in canonical form, and the resulting generated sentence.

FlowDoc (Passonneau et al., 1996) takes as input flow diagrams representing the structure and operations of a business unit, either as it is currently operating or after a proposed re-organization. Like PLANDoc, it interfaces with another tool developed at Bellcore, ShowBiz, which maintains the graphical representation and allows the exploration of possible alternatives by the re-engineering consultant. The diagrams resulting from re-engineering analysis are quite complex, with numerous nodes annotated with a large number of attributes. FlowDoc identifies the core components, participants, and actions of each flow diagram and produces a short textual summary. Figure 2 shows an example input flow diagram, the representation of a sample node in that diagram as presented to FlowDoc by ShowBiz, FlowDoc's description of the same in-

```
RUNID fiberall FIBER
6/19/93 act yes
BFA 1301 2 1995
FA 1201 2 1995
FA 1401 2 1995
FA 1501 2 1995
ANF CO 1103 2 1995 48
ANF 1201 1301 2 1995 24
ANF 1401 1501 2 1995 24
END 856.0 670.2
```

(a) Fragment of LEIS-PLAN's execution trace for a particular plan.

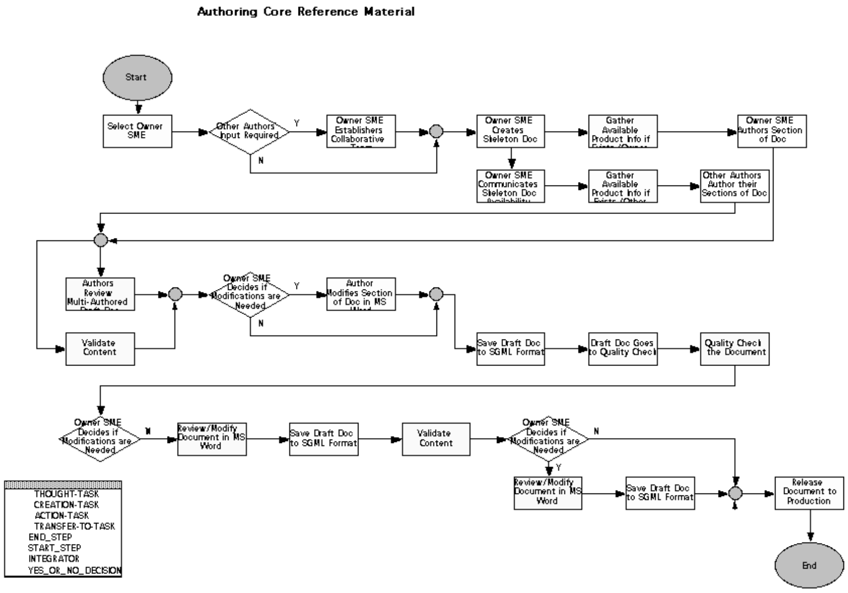
```
((cat domain_message)
 (admin ((msg-num 19)
         (runid FIBERALL)
         (prev-runid ALLDLC)
         (status act)
         (saved yes)))
 (class refinement)
 (ref-type FIBER)
 (action ACTIVATION)
 (csa-site 1301)
 (date ((year 1995) (quarter 2))))
```

(b) PLANDoc's representation of (a) in canonical form.

```
RUN-ID FIBERALL demanded that PLAN
activate fiber for CSAs 1201, 1301,
1401 and 1501 in 1995 Q2.
```

(c) Sentence generated by PLANDoc from the data in (b) and three other similar messages.

Figure 1: Sample Input, Canonical Representation, and Output of PLANDoc.



(a) Input flow diagram.

```
(make-flownode 26 'thought-task
   :node-position 32899435
   :who 'SME
   :does_what "review"
   :to_whom_or_what
       'draft_document_in_MS_Word_format)
```

(b) Input representation of a sample node.

```
((cat domain_msg)
 (msg_id 14)
 (msg-class salient-task)
 (workflow_id 000931)
 (activity-class thought_activity)
 (does_what review)
 (to_whom_or_what ms_word_doc)
 (count 3))
```

(c) FlowDoc's canonical representation of the information in (b), aggregated with information from two other similar nodes.

```
The most frequent tasks in this workflow are those of creating,
reviewing and saving documents.
```

(d) Generated sentence from the canonical message in (c) and from similar messages corresponding to other frequent tasks in the input diagram.

Figure 2: Sample Input Flow Diagram, Input Description of a Single Node, Canonical Representation of a Set of Nodes after Aggregation, and Corresponding Generated Sentence for FlowDoc.

```
select host, count(host)
  from zeddoc_view
  where (date_time between '01-JAN-95' and '31-DEC-96')
```

(a) Fragment of SQL query automatically generated by ZEDDoc's database query subsystem.

HOST	COUNT(HOST)
santos.doc.ic.ac.uk	12
896ed78a.extern.ucsd.edu	7
thor.dai.ed.ac.uk	7
hvlassar.port.net	6
vip-b.enel.ucalgary.ca	4
baugi.ifi.uio.no	3
pm2-05.sundial.net	3
194.80.129.254	2
abest206.abest.com	2
...	

(b) Part of the database output for the query in (a).

```
((msg-class user-domain)
 (santos.doc.ic.ac.uk 12.0)
 (896ed78a.extern.ucsd.edu 7.0)
 (thor.dai.ed.ac.uk 7.0)
 (hvlassar.port.net 6.0)
 (vip-b.enel.ucalgary.ca 4.0)
 (baugi.ifi.uio.no 3.0)
 (pm2-05.sundial.net 3.0)
 (other 2.0)
 (abest206.abest.com 2.0)
 ...)
```

(c) ZEDDoc's canonical representation of the information in (b).

```
For the ads of interest, the most frequent Internet user domains were European
Internet domains at 28 percent and U.S. network domains at 23 percent.
```

(d) One of the sentences generated by ZEDDoc from the full information about network hosts, which is partially shown in (c).

Figure 3: Automatically Generated SQL Query, Partial Database Output, Corresponding Canonical Representation, and one of the Corresponding Sentences Produced by ZEDDoc.

formation aggregated over several similar nodes in the diagram, and the sentence generated to express this information.

ZEDDoc summarizes the underlying ZED application's WWW activity. ZED manages a database of advertisement images to satisfy Web advertising contracts.[1] It selects ads to display in predefined slots in a manner that optimizes the satisfaction of the advertising contracts. Whenever ZED displays a Web page, it determines what ads to display and creates database entries for each displayed ad. ZEDDoc integrates a browser, the summary generator, and ZED's Oracle™ database of WWW transactions in a client-server architecture. By accessing the transaction database, ZEDDoc can produce short summaries of ad activity within a user-specified time frame for a user-specified set of ads. Summaries contain, for example, demographic generalizations pertaining to potentially large numbers of hits. An example of ZEDDoc's input, internal representation, and output is shown in Figure 3.

[1]Zed has evolved into a product, the Adapt/X Advertiser™.

3 A Common Architecture

While PLANDoc, FlowDoc, and ZEDDoc all share a common foundation, they embody distinctly different text generation applications. However, we aimed during the design of both FlowDoc and ZEDDoc to utilize as much of PLANDoc's architecture as possible, often adapting and generalizing modules that were originally written with only the PLANDoc system in mind.

All three systems employ a modular pipeline architecture. A pipeline architecture is one that separates the functions involved in text generation, such as content planning, discourse organization, lexicalization, and syntactic realization, into distinct modules that operate in sequence. Modular pipeline architectures have a long history of use in text generation systems (Kukich, 1983a; McKeown, 1985; McDonald and Pustejovsky, 1986; Reiter, 1994), although recent work argues for the need for interaction between modules (Danlos, 1987; Rubinoff, 1992; McKeown et al., 1993). The most powerful argument for using pipeline architectures is the potential benefit of re-using individual modules for subsequent applications. However, with the exception of surface realization modules such as FUF/SURGE (Elhadad, 1992; Robin, 1994), actual code re-use has been minimal due to the lack of agreement about the order and grouping of subprocesses into modules.

In PLANDoc, FlowDoc, and ZEDDoc, we utilize the following main modules, in the order listed below:

- **Message Generator**: The message generator transcribes the raw data from LEIS-PLAN execution traces, ShowBiz, or ZED transaction logs into instances of *message classes*. We refer to simple collections of (possibly nested) attribute-value pairs pertaining to a single event as *messages*. Message classes are domain-specific (e.g., there are 30 of them in PLANDoc, 13 in FlowDoc, and 6 in ZEDDoc), but they all share the same representation as the basic content unit. In all three systems, generalization must occur at this level in order to create semantically concise messages from relatively large amounts of input data.

- **Ontologizer**: In PLANDoc, a pipelined ontologizer enriches messages with domain-specific knowledge that is not explicitly present in the input. In FlowDoc and ZEDDoc, semantic enrichment is done at various stages by consulting external ontologies.

- **Discourse Organizer**: The discourse organizer performs all the remaining functions prior to lexicalization and surface generation[2]. Three sub-modules apply general discourse coherence constraints at the levels of discourse, sentence, and sentence constituent. The first module performs aggregation and text linearization operations using an ontology of rhetorical predicates derived from Hobbs (1985) and Polanyi (1988). Linear order and prominence of the subconstituents are then determined, followed by constraints on subconstituents that affect lexical choice (e.g., centering and informational constraints, as in (Passonneau, 1996)).

- **Lexicalizer**: The lexicalizer maps message attributes into thematic/case roles, and chooses appropriate content (open-class) words for the values of these attributes.

- **Surface Generator**: This module maps thematic roles into syntactic roles and builds syntactic constituents, chooses function (closed-class) words, ensures grammatical agreement, and linearizes to produce the final surface sentence.

Our message generator modules are largely domain-specific, and we have made major changes to them while porting them to new applications. Even so, their ontological generalization technique, which produces semantically concise descriptions from frequency data, is domain-independent. Our final surface generation module is completely domain-independent; it employs the FUF/SURGE (Elhadad, 1991; Robin, 1994) text generation tools, and was re-used in all three systems with virtually no modifications. Modules near the middle of the pipeline provide the most interesting examples of code that *can* be re-used if it is general enough and relies on plug-and-play knowledge bases rather than hard-coded data. We return to this issue of code re-use and of the evolution of our modules to accommodate it in Section 5.

4 A Common Representation

All three systems employ a consistent, standardized attribute-value data format that persists from each module to the next. Examples of this internal data format were shown in Figures 1–3. This format is used for representing and processing conceptual-semantic, lexical-semantic, syntactic, and other linguistic information. Its persistent use facilitates inter-module communication and module independence, hence re-usability. Furthermore, it does not restrict the kinds of information that can be represented, and it is common to many non-NLP computational systems and languages (e.g., relational databases), thus making it easier for text generation systems to interface with existing applications.

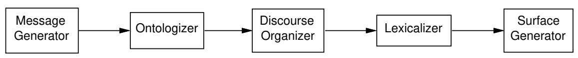
The input to each of our three systems came from very different sources, some closer than others to attribute-value message format. PLANDoc's input came from n-tuple records representing program execution traces, so it required a filter to transform it into messages. FlowDoc's input came from ASCII representations of nodes and links in work flow diagrams which were already essentially in attribute-value format. ZEDDoc's input, representing Web

[2] In previous work we referred to this module as the Sentence Planner (Passonneau et al., 1996).

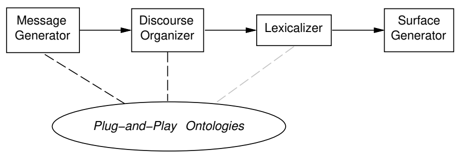
activity data, had been stored in an Oracle™ relational database by its application, so it too required little transformation.

5 Architectural Evolution

As discussed earlier, a practical goal for text generation research is to converge on a separation of functions into modules that can be independently re-used. Towards this goal, we have generalized and refined our architecture with each successive application. In fact, we significantly adapted our PLANDoc architecture for use in FlowDoc, but we were able to re-use the FlowDoc architecture and much of its code in ZEDDoc. Figure 4 contrasts the architecture of PLANDoc with those of FlowDoc and ZEDDoc.



(a) Overall architecture for PLANDoc.



(b) Overall architecture for FlowDoc and ZEDDoc.

Figure 4: Contrasting the Architecture of the Three Text Generation Systems.

The obvious architectural change from PLANDoc to FlowDoc (and ZEDDoc) is the extraction of ontological knowledge from the processing pipeline. Ontological knowledge is necessarily domain-specific, so this modification allowed us to implement significantly more general Message Generation and Discourse Organization modules and a somewhat more general Lexicalization module. These more general modules rely on external knowledge bases to supply the domain-specific information that was previously embedded in the code. Thus, we can replace the external knowledge base when moving to a new domain or application without having to modify the module itself. One of our future research goals is to further extract domain-specific lexical knowledge and further generalize the lexicalizer module (Jing et al., 1997).

What is not so obvious from Figure 4 are the consistencies and shifts in function among the modules. In fact, the functions of the Lexicalization and Surface Generation modules remained constant across all three systems. But the functions of the first three modules shifted significantly from PLANDoc to FlowDoc. In particular, the function of message aggregation lay exclusively in the Discourse Organization module in PLANDoc (Shaw, 1995), whereas aggregation functions are executed in both the Message Generation and Discourse Organization modules in FlowDoc.

Because the development of domain-independent, plug-and-play ontology modules is one of the major features that affected these shifts in function, and because such modules greatly increase the portability of the system, we devote the next section to a more detailed description of the function of ontological generalization.

6 Ontological Generalization

Ontological generalization refers to the problem of composing, with the help of an ontology, a concise description for a multi-set of concepts. For example, FlowDoc's output sentence shown in Figure 2

```
The most frequent tasks in this
workflow are those of creating,
reviewing and saving documents.
```

concisely describes a multi-set of ten specific task nodes in the flow diagram by locating superclass concepts in the ontology that encompass the specific predicates and objects of the task nodes. Our aim is to compose a description that is concise without sacrificing much in accuracy.

While PLANDoc made extensive use of conjunction, ellipsis, and paraphrasing to produce a concise summary, ontological relations were not heavily used. For FlowDoc we implemented a more general, domain-independent solution. We were able to re-use this module with minor modifications in ZEDDoc, after replacing the ontological knowledge base.

Our ontological generalization algorithm works as follows. Given a set $C_O = \{O_1, O_2, \ldots, O_N\}$ of objects of a given predicate-class and an associated list $(c_1, c_2, \ldots, c_N)$ of their occurrence counts, we compute an optimal set of concept generalizations $\{G_1, G_2, \ldots, G_M\}$ such that each generalization replaces a subset of C_O while maintaining a reasonable trade-off between the accuracy, specificity, and verbosity of the resulting description.

We consider as candidate concept generalizations the actual members of C_O and all the concepts in the domain ontology that subsume one or more of them. Each such candidate concept generalization is evaluated on its suitability to replace a given subset of C_O using a weighted sum formula, trading-off along two antagonistic dimensions:

- **Coverage**, measuring how many of the objects in the subset (proportionally weighted according to their occurrence counts c_i) are actually subsumed by the candidate generalization.

- **Specificity**, defined as the average semantic distance between each element of the subset and the candidate generalization.

The semantic distance currently used is simply the number of levels between each object and the generalization in the domain ontology. It could be easily changed to an information-based distance, e.g., along the lines of the metrics proposed in (Resnik, 1995), who measures semantic distance between two concepts as a function of the lexical probabilities of their common superclasses.

To compute the optimal set of generalizations, the algorithm starts by generating all possible partitions of the given set of objects[3], then locates the best single-term description for each subset in the partition by applying the procedure outlined above to each candidate generalization, and finally combines the single-term description scores in one number. The final score is adjusted by two additional penalties:

- A *verbosity* penalty, penalizing descriptions with more than one generalization (exponentially more as the number of terms in the description increases).

- A *heterogeneity* penalty, for descriptions that are locally optimal but significantly lower in the ontology (more specific) than the global specificity level.

The global specificity level indicates the appropriate overall level of detail. It is computed by applying the above ontological generalization procedure to the collection of *all* the objects appearing in the input graph, across all actions. It implements the idea of "basic level" descriptions from (Rosch, 1978) for the application domain modeled by the work flow. For example, while processing a flow diagram which covers documents of many types, our algorithm will have a bias in favor of the generic term "Document" rather the too-specific term "Draft document in SGML format"; a trade-off between the heterogeneity penalty and other components of the description score occurs if the latter term looks locally optimal.

[3]With some performance-imposed constraints, since the number of possible partitions grows exponentially with the number of objects and the number of subsets in the partition.

The same generalization method for sets of ⟨concept, occurrence count⟩ pairs was applied in ZEDDOC, but instead of actions or graph components, the concepts were Internet addresses or ZED page types. ZED requires semantic types to be assigned to WWW pages and ads to help determine which ads from its database can be inserted in predefined ad slots. When a ZEDDOC user requests a summary of activity pertaining to a particular set of ads for a given time period, the raw data consists in part of frequency lists indicating how many users from a given Internet node saw the relevant ads and how many of the displayed pages corresponded to particular semantic types. One minor change for ZEDDOC was the replacement of predefined absolute frequency thresholds for determining the salience of items with relative ones.

To summarize the Internet domain or page type data, ZEDDOC relies on plug-and-play ontologies. Specialization subtrees rooted at certain concepts, e.g., the Internet domain, can be replaced so long as at least one lexicalization is provided for every concept. Our ontology for the Internet domain combined world knowledge with the implicit hierarchical structure of domain names. For example, through hand analysis of WWW logs we created a geographical categorization of university nodes, on the assumption that such demographic information is important to advertisers.

7 Component Re-Use Revisited

The major theme throughout this paper has been how we re-used components from our original PlanDoc system to implement the subsequent FLOWDOC and ZEDDOC systems, significantly cutting development time. In this section, we summarize our experiences regarding code re-use.

- The message generator offers limited possibilities for reuse because it directly interfaces to an application-specific external source. Limited code sharing was possible however, because of our choice of a common representation format for all three systems.

- As noted briefly in Section 3, the FLOWDOC architecture had distinct modules pertaining to the three levels of discourse, sentence, and sentence constituent. Retaining this more general architecture in ZEDDOC proved useful with respect to one additional required functionality, namely the ability to produce plain text or

HTML output. The three levels of discourse organization were exploited in ZEDDoc primarily to distinguish between HTML commands that pertain to the overall layout (e.g., paragraph divisions) versus those that pertain to sentence-internal features (e.g., fonts).

- At the lexicalization level, we achieved only partial generalization of the lexicalizer's code. Given the state of the art in natural language generation, the lexicon remains necessarily domain-specific. However, we are exploring ways to remove domain-specific lexical knowledge from the system pipeline, as we did with domain-specific ontological and discourse knowledge.

  We are building a large-scale general lexicon for generation, which provides syntactic and partial semantic knowledge and can be used to select the generated sentence structure and possible paraphrases (Jing et al., 1997). By using this general lexicon together with a smaller domain-specific lexicon or with information extracted from a corpus from the application domain, we expect to significantly simplify the development of the lexicalization module, improving its reliability and portability.

- At the final surface generation level, we took advantage of prior progress in component standardization and used FUF (Functional Unification Formalism) and its corresponding extensive English surface grammar SURGE. As a result, the surface generation module was ported unchanged to the other systems.

8 Conclusion

By teasing apart some of PLANDoc's modules and partially re-configuring others, we were able to port our text generation system to two completely new domains, those of flow chart and WWW activity summarization. In the process, we devised domain-independent message aggregation and discourse restructuring modules for FlowDoc that we re-used intact for ZEDDoc. Indeed, we believe that our ontological generalization algorithm (i.e., message aggregation guided by quantitative formulas over plug-and-play ontologies) is generally domain-independent. We are exploring ways to introduce probability estimates in our weighting functions for message aggregation, linking the static ontology with corpus-observable variations in concept use and coverage.

Re-usable tools and techniques can provide leverage for building practical text generation applications. They can also facilitate research leading to increasingly more general and more useful tools. This has been our experience in implementing the three text generation systems covered in this paper which are all based on a common architecture, a common representation format, and a common, evolving foundation of text generation tools.

At least three other factors that are critical to practical and commercial success should be mentioned though we cannot discuss them here. Two of them, i) extensive user-needs analysis and feedback and ii) target corpus compilation and analysis, are highly correlated with the relative success of each of our systems. These two factors are discussed in more detail in previous papers (Kukich et al., 1994; Kukich, 1983b). A third, undocumented factor, the rigorous pre-release testing of the system under conditions similar to its deployment environment, played a critical role in PLANDoc's success.

Acknowledgment

Research on these projects at Columbia University was supported by grants from Bellcore.

References

Laurence Danlos. 1987. *The Linguistic Basis of Text Generation.* Studies in Natural Language Processing. Cambridge University Press, Cambridge, England.

Michael Elhadad. 1991. FUF user manual — version 5.0. Technical Report CUCS-038-91, Department of Computer Science, Columbia University.

Michael Elhadad. 1992. *Using Argumentation to Control Lexical Choice: A Functional Unification-Based Approach.* Ph.D. thesis, Department of Computer Science, Columbia University.

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

Hongyan Jing, Kathleen McKeown, and Rebecca Passonneau. 1997. Building a rich large-scale lexical base for generation. Technical Report CUCS-016-97, Department of Computer Science, Columbia University.

Karen Kukich, Kathleen McKeown, James Shaw, Jacques Robin, Jong Lim, Neal Morgan, and Jim

Phillips. 1994. User needs analysis and design methodology for an automated documentation generator. In Antonio Zampolli, Nicoletta Calzolari, and Martha Palmer, editors, *Current Issues in Computational Linguistics: In Honour of Don Walker*, pages 109–115. Kluwer Academic Press, Boston, Massachussets.

Karen Kukich. 1983a. Design and implementation of a knowledge-based text generator. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pages 145–150, Cambridge, Massachusetts, June.

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

David McDonald and James Pustejovsky. 1986. Description-directed natural language generation. In *Proceedings of the 9th International Joint Conference on Artifical Intelligence*, pages 799–805, Los Angeles, California.

Kathleen McKeown, Jacques Robin, and Michael Tanenblatt. 1993. Tailoring lexical choice to the user's vocabulary in multimedia explanation generation. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 226–234, Columbus, Ohio, June.

Kathleen McKeown, Karen Kukich, and James Shaw. 1994. Practical issues in automatic documentation generation. In *Proceedings of the 1994 Applied Natural Language Processing Conference*, pages 7–14, Stuttgart, Germany, October.

Kathleen McKeown. 1985. The need for text generation. Technical Report CUCS-173-85, Department of Computer Science, Columbia University.

Rebecca Passonneau, Karen Kukich, Jacques Robin, Vasileios Hatzivassiloglou, Larry Lefkowitz, and Hongyan Jing. 1996. Generating summaries of work flow diagrams. In *Proceedings of the International Conference on Natural Language Processing and Industrial Applications*, pages 204–210, New Brunswick, Canada, June. University of Moncton.

Rebecca Passonneau, Karen Kukich, Kathleen McKeown, Dragomir Radev, and Hongyan Jing. 1997. Summarizing web traffic: A portability exercise. Technical Report CUCS-009-97, Department of Computer Science, Columbia University.

Rebecca Passonneau. 1996. Using centering to relax Gricean informational constraints on discourse anaphoric noun phrases. *Language and Speech*, **39**(2–3):229–265, April–September. Special double issue on Discourse and Syntax, edited by Judy Delin and Jon Oberlander.

Livya Polanyi. 1988. A formal model of discourse structure. *Journal of Pragmatics*, **12**:601–638.

Ehud Reiter. 1994. Has a consensus NL generation architecture appeared, and is it psycholinguistically plausible? In *Proceedings of the 1994 International Natural Language Generation Workshop*, pages 163–170, Kennebunkport, Maine.

Philip Resnik. 1995. Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, volume 1, pages 448–453, Montréal, Québec, Canada, August. Morgan Kaufmann, San Mateo, California.

Jacques Robin. 1994. *Revision-Based Generation of Natural Language Summaries Providing Historical Background: Corpus-Based Analysis, Design, Implementation, and Evaluation.* Ph.D. thesis, Department of Computer Science, Columbia University. Also Technical Report CU-CS-034-94.

Eleanor Rosch. 1978. Principles of categorization. In Eleanor Roschand and Barbara B. Lloyd, editors, *Cognition and Categorization*, pages 27–48. Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Robert Rubinoff. 1992. A cooperative model of strategy and tactics in generation. In Robert Dale, Eduard Hovy, Dietmar Röesner, and Oliviero Stock, editors, *Aspects of Automated Natural Language Generation.* Springer Verlag. Presented at the 6th International Workshop on Natural Language Generation, Trento, Italy.

James Shaw. 1995. Conciseness through aggregation in text generation. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (Student Session)*, pages 329–331, June.