AN OVERVIEW OF THE NIGEL TEXT GENERATION GRAMMAR [1]

William C. Mann
USC/Information Sciences Institute
4676 Admiralty Way # 1101
Marina del Rey, CA 90291

Abstract

Research on the text generation task has led to creation of a large systemic grammar of English, Nigel, which is embedded in a computer program. The grammar and the systemic framework have been extended by addition of a semantic stratum. The grammar generates sentences and other units under several kinds of experimental control.

This paper describes augmentations of various precedents in the systemic framework. The emphasis is on developments which control the text to fulfill a purpose, and on characteristics which make Nigel relatively easy to embed in a larger experimental program.

1 A Grammar for Text Generation - The Challenge

Among the various uses for grammars, text generation at first seems to be relatively new. The organizing goal of text generation, as a research task, is to describe how texts can be created in fulfillment of text needs.[2]

Such a description must relate texts to needs, and so must contain a functional account of the use and nature of language, a very old goal. Computational text generation research should be seen as simply a particular way to pursue that goal.

As part of a text generation research project, a grammar of English has been created and embodied in a computer program. This grammar and program, called Nigel, is intended as a component of a larger program called Penman. This paper introduces Nigel, with just enough detail about Penman to show Nigel's potential use in a text generation system.

1.1 The Text Generation Task as a Stimulus for Grammar Design

Text generation seeks to characterize the use of natural languages by developing processes (computer programs) which can create appropriate, fluent text on demand. A representative research goal would be to create a program which could write a text that serves as a commentary on a game transcript, making the events of the game understandable.[3]

The guiding aims in the ongoing design of the Penman text generation program are as follows:

1. To learn, in a more specific way than has previously been achieved, how appropriate text can be created in response to text needs.

2. To identify the dominant characteristics which make a text appropriate for meeting its need.

3. To develop a demonstrable capacity to create texts which meet some identifiable practical class of text needs.

Seeking to fill these goals, several different grammatical frameworks were considered. The systemic framework was chosen, and it has proven to be an entirely agreeable choice. Although it is relatively unfamiliar to many American researchers, it has a long history of use in work on concerns which are central to text generation. It was used by Winograd in the SHRDLU system, and more extensively by others since [Winograd 72, Davey 79, McKeown 82, McDonald 80]. A recent state of the art survey identifies the systemic framework as one of a small number of linguistic frameworks which are likely to be the basis for significant text generation programs in this decade [Mann 82a].

One of the principal advantages of the systemic framework is its strong emphasis on "functional" explanations of grammatical phenomena. Each distinct kind of grammatical entity is associated with an expression of what it does for the speaker, so that the grammar indicates not only what is possible but why it would be used. Another is its emphasis on principled, justified descriptions of the ***choices*** which the grammar offers, i.e. all of its optionality. Both of these emphases support text generation programming significantly. For these and other reasons the systemic framework was chosen for Nigel.

Basic references on the systemic framework include: [Berry 75, Berry 77, Halliday 76a, Halliday 76b, Hudson

[1] This research was supported by the Air Force Office of Scientific Research contract No. F49620-79-C-0181. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research of the U.S. Government.

[2] A text need is the earliest recognition on the part of the speaker that the immediate situation is one in which he would like to produce speech. In this report we will alternate freely between the terms speaker, writer and author, between hearer and reader, and between speech and text. This is simply partial accommodation of prevailing jargon; no differences are intended.

[3] This was accomplished in work by Anthony Davey [Davey 79]; [McKeown 82] is a comparable more recent study in which the generated text described structural and definitional aspects of a data base.

76, Halliday 81, de Joia 80, Fawcett 80].[4]

1.2 Design Goals for the Grammar

Three kinds of goals have guided the work of creating Nigel.

1. To specify in total detail how ***the systemic framework*** can generate syntactic units, using the computer as the medium of experimentation.

2. To develop a ***grammar of English*** which is a good representative of the systemic framework and useful for demonstrating text generation on a particular task.

3. To specify how the grammar can be ***regulated effectively by the prevailing text need*** in its generation activity.

Nigel is intended to serve not only as a part of the Penman system, but also eventually as a portable generational grammar, a component of future research systems investigating and developing text generation.

Each of the three goals above has led to a different kind of activity in developing Nigel and a different kind of specification in the resulting program, as described below. The three design goals have not all been met, and the work continues.

1. Work on the first goal, specifying the framework, is essentially finished (see section 2.1). The Interlisp program is stable and reliable for its developers.

2. Very substantial progress has been made on creating the grammar of English; although the existing grammar is apparently adequate for some text generation tasks, some additions are planned.

3. Progress on the third goal, although gratifying, is seriously incomplete. We have a notation and a design method for relating the grammar to prevailing text needs, and there are worked out examples which illustrate the methods the demonstration paper in [Mann 83](see section 2.3.)

2 A Grammar for Text Generation - The Design

2.1 Overview of Nigel's Design

The creation of the Nigel program has required evolutionary rather than radical revisions in systemic notation, largely in the direction of making well-precedented ideas more explicit or detailed. Systemic notation deals principally with three kinds of entities: 1) systems, 2) realizations of systemic choices (including function structures), and 3) lexical items. These three account for most of the notational devices, and the Nigel program has separate parts for each.

[4]This work would not have been possible without the active participation of Christian Matthiessen, and the participation and past contributions of Michael Halliday and other systemicists.

Comparing the systemic functional approach to a structural approach such as context-free grammar, ATNs or transformational grammar, the differences in style (and their effects on the programmed result) are profound. Although it is not possible to compare the approaches in depth here, we note several differences of interest to people more familiar with structural approaches:

1. Systems, which are most like structural rules, do not specify the order of constituents. Instead they are used to specify sets of features to be possessed by the grammatical construction as a whole.

2. The grammar typically pursues several independent lines of reasoning (or specification) whose results are then combined. This is particularly difficult to do in a structurally oriented grammar, which ordinarily expresses the state of development of a unit in terms of categories of constituents.

3. In the systemic framework, all variability of the structure of the result, and hence all grammatical control, is in one kind of construct, the system. In other frameworks there is often variability from several sources: optional rules, disjunctive options within rules, optional constituents, order of application and so forth. For generation these would have to be coordinated by methods which lie outside of the grammar, but in the systemic grammar the coordination problem does not exist.

2.1.1 Systems and Gates

Each ***system*** contains a set of alternatives, symbols called ***grammatical features***. When a system is ***entered***, exactly one of its grammatical features must be chosen. Each system also has an ***input expression***, which encodes the conditions under which the system is entered.[5] During the generation, the program keeps track of the ***selection expression***, the set of features which have been chosen up to that point. Based on the selection expression, the program invokes the realization operations which are associated with each feature chosen.

In addition to the systems there are ***Gates***. A gate can be thought of as an input expression which activates a particular grammatical feature, without choice.[6] These grammatical features are used just as those chosen in systems. Gates are most often used to perform realization in response to a collection of features.[7]

[5]Input expressions are Boolean expressions of features, without negation, i.e. they are composed entirely of feature names, together with *And*, *Or* and parentheses. (See the figures in the demonstration paper in [Mann 83] for examples.)

[6]See the figure entitled Transitivity I in [Mann 83] for examples and further discussion of the roles of gates.

[7]Each realization operation is associated with just one feature, there are no realization operations which depend on more than one feature, and no rules corresponding to Hudson's function realization rules. The gates facilitate eliminating this category of rules, with a net effect that the notation is more homogeneous.

2.1.2 Realization Operators

There are three groups of realization operators: those that build structure (in terms of grammatical functions), those that constrain order, and those that associate features with grammatical functions.

1. The realization operators which build structure are ***Insert***, ***Conflate***, and ***Expand***. By repeated use of the structure building functions, the grammar is able to construct sets of ***function bundles***, also called ***fundles***. None of them are new to the systemic framework.

2. Realization operators which constrain order are ***Partition***, ***Order***, ***OrderAtFront*** and ***OrderAtEnd***. Partition constrains one function (hence one fundle) to be realized to the left of another, but does not constrain them to be adjacent. Order constrains just as Partition does, and in addition constrains the two to be realized adjacently. OrderAtFront constrains a function to be realized as the leftmost among the daughters of its mother, and OrderAtEnd symmetrically as rightmost. Of these, only Partition is new to the systemic framework.

3. Some operators associate features with functions. They are ***Preselect***, which associates a grammatical feature with a function (and hence with its fundle); ***Classify***, which associates a ***lexical feature*** with a function; ***OutClassify***, which associates a lexical feature with a function in a preventive way; and ***Lexify***, which forces a particular lexical item to be used to realize a function. Of these, OutClassify and Lexify are new, taking up roles previously filled by Classify. OutClassify restricts the realization of a function (and hence fundle) to be a lexical item which does not bear the named feature. This is useful for controlling items in exception categories (e.g. reflexives) in a localized, manageable way. Lexify allows the grammar to force selection of a particular item without having a special lexical feature for that purpose.

In addition to these realization operators, there is a set of ***Default Function Order Lists***. These are lists of functions which will be ordered in particular ways by Nigel, provided that the functions on the lists occur in the structure, and that the realization operators have not already ordered those functions. A large proportion of the constraint of order is performed through the use of these lists.

The realization operations of the systemic framework, especially those having to do with order, have not been specified so explicitly before.

2.1.3 The Lexicon

The lexicon is defined as a set of arbitrary symbols, called ***word names***, such as "builten", associated with symbols called ***spellings***, the lexical items as they appear in text. In order to keep Nigel simple during its early development, there is no formal provision for morphology or for relations between items which arise from the same root.

Each word name has an associated set of ***lexical features***.

Lexify selects items by word name; Classify and OutClassify operate on sets of items in terms of the lexical features.

2.2 The Grammar and Lexicon of English

Nigel's grammar is partly based on published sources, and is partly new. It has all been expressed in a single homogeneous notation, with consistent naming conventions and much care to avoid reusing names where identity is not intended. The grammar is organized as a single network, whose one entry point is used for generating every kind of unit.[8]

Nigel's lexicon is designed for test purposes rather than for coverage of any particular generation task. It currently recognizes 130 lexical features, and it has about 2000 lexical items in about 580 distinct categories (combinations of features).

2.3 Choosers - The Grammar's Semantics

The most novel part of Nigel is the semantics of the grammar. One of the goals identified above was to "specify how the grammar can be regulated effectively by the prevailing text need." Just as the grammar and the resulting text are both very complex, so is the text need. In fact, grammar and text complexity actually reflect the prior complexity of the text need which gave rise to the text. The grammar must respond selectively to those elements of the need which are represented by the unit being generated at the moment.

Except for lexical choice, all variability in Nigel's generated result comes from variability of choice in the grammar. Generating an appropriate structure consists entirely in making the choices in each system appropriately. The semantics of the grammar must therefore be a semantics of choices in the individual systems; the choices must be made in each system according to the appropriate elements of the prevailing need.

In Nigel this semantic control is localized to the systems themselves. For each system, a procedure is defined which can declare the appropriate choice in the system. When the system is entered, the procedure is followed to discover the appropriate choice. Such a procedure is called a ***chooser*** (or "choice expert".) The chooser is the semantic account of the system, the description of the circumstances under which each choice is appropriate.

To specify the semantics of the choices, we needed a notation for the choosers as procedures. This paper describes that notation briefly and informally. Its use is exemplified in the Nigel demonstration [Mann 83] and developed in more detail in another report [Mann 82b].

To gain access to the details of the need, the choosers must in some sense ask questions about particular entities. For example, to decide between the grammatical features Singular and Plural in creating a NominalGroup, the Number chooser (the

---

[8] At the end of 1982, Nigel contained about 220 systems, with all of the necessary realizations specified. It is thus the largest systemic grammar in a single notation, and possibly the largest grammar of a natural language in any of the functional linguistic traditions. Nigel is programmed in INTERLISP.

chooser for the Number system, where these features are the options) must be able to ask whether a particular entity (already identified elsewhere as the entity the NominalGroup represents) is unitary or multiple. That knowledge resides outside of Nigel, in the ***environment.***

The environment is regarded informally as being composed of three disjoint regions:

1. The ***Knowledge Base***, consisting of information which existed prior to the text need;

2. The ***Text Plan***, consisting of information which was created in response to the text need, but before the grammar was entered;

3. The ***Text Services***, consisting of information which is available on demand, without anticipation.

Choosers must have access to a stock of symbols representing entities in the environment. Such symbols are called ***hubs***. In the course of generation, hubs are associated with grammatical functions; the associations are kept in a ***Function Association Table***, which is used to reaccess information in the environment. For example, in choosing pronouns the choosers will ask questions about the multiplicity of an entity which is associated with the THING function in the Function Association Table. Later they may ask about the gender of the same entity, again accessing it through its association with THING. This use of grammatical functions is an extension of previous uses. Consequently, relations between referring phrases and the concepts being referred to are captured in the Function Association Table. For example, the function representing the NominalGroup as a whole is associated with the hub which represents the thing being referred to in the environment. Similarly for possessive determiners, the grammatical function for the determiner is associated with the hub for the possessor.

It is convenient to define choosers in such a way that they have the form of a tree. For any particular case, a single path of operations is traversed. Choosers are defined principally in terms of the following operations:

1. ***Ask*** presents an inquiry to the environment. The inquiry has a fixed predetermined set of possible responses, each corresponding to a branch of the path in the chooser.

2. ***Identify*** presents an inquiry to the environment. The set of responses is open-ended. The response is put in the Function Association Table, associated with a grammatical function which is given (in addition to the inquiry) as a parameter to the Identify operator.[9]

3. ***Choose*** declares a choice.

4. ***CopyHub*** transfers an association of a hub from one grammatical function to another.[10]

Choosers obtain information about the immediate circumstances in which they are generating by presenting ***inquiries*** to the environment. Presenting inquiries, and receiving replies constitute the only way in which the grammar and its environment interact.

An inquiry consists of an ***inquiry operator*** and a sequence of ***inquiry parameters***. Each inquiry parameter is a grammatical function, and it represents (via the Function Association Table) the entities in the environment which the grammar is inquiring about. The operators are defined in such a way that they have both formal and informal modes of expression. Informally, each inquiry is a predefined question, in English, which represents the issue that the inquiry is intended to resolve for any chooser that uses it. Formally, the inquiry shows how systemic choices depend on facts about particular grammatical functions, and in particular restricts the account of a particular choice to be responsive to a well-constrained, well-identified collection of facts. Both the informal English form of the inquiry and the corresponding formal expression are regarded as parts of the semantic theory expressed by the choosers which use the inquiry. The entire collection of inquiries for a grammar is a definition of the semantic scope to which the grammar is responsive at its level of delicacy.

Figure 1 shows the chooser for the ProcessType system, whose grammatical feature alternatives are Relational, Mental, Verbal and Material.

Notice that in the ProcessType chooser, although there are only four possible choices, there are five paths through the chooser from the starting point at the top, because Mental processes can be identified in two different ways: those which represent states of affairs and those which do not. The number of termination points of a chooser often exceeds the number of choices available.

Table 1 shows the English forms of the questions being asked in the ProcessType chooser. (A word in all capitals names a grammatical function which is a parameter of the inquiry.)

Table 1: English Forms of the Inquiry Operators for the ProcessType Chooser

StaticConditionQ	Does the process PROCESS represent a static condition or state of being?
VerbalProcessQ	Does the process PROCESS represent symbolic communication of a kind which could have an addressee?
MentalProcessQ	Is PROCESS a process of comprehension, recognition, belief, perception, deduction, remembering, evaluation or mental reaction?

The sequence of inquiries which the choosers present to the environment, together with its responses, creates a dialogue. The unit generated can thus be seen as being formed out of a negotiation between the choosers and the environment. This is a particularly instructive way to view the grammar and its semantics, since it identifies clearly what assumptions are being made and what dependencies there are between the unit and the environment's representation of the text need. (This is the kind of dialogue represented in the demonstration paper in [Mann 83].)

[9] See the demonstration paper in [Mann 83] for an explanation and example of its use

[10] There are three others which have some linguistic significance: Pledge, TermPledge, and ChoiceError. These are necessary but do not play a central role. They are named here just to indicate that the chooser notation is very simple.

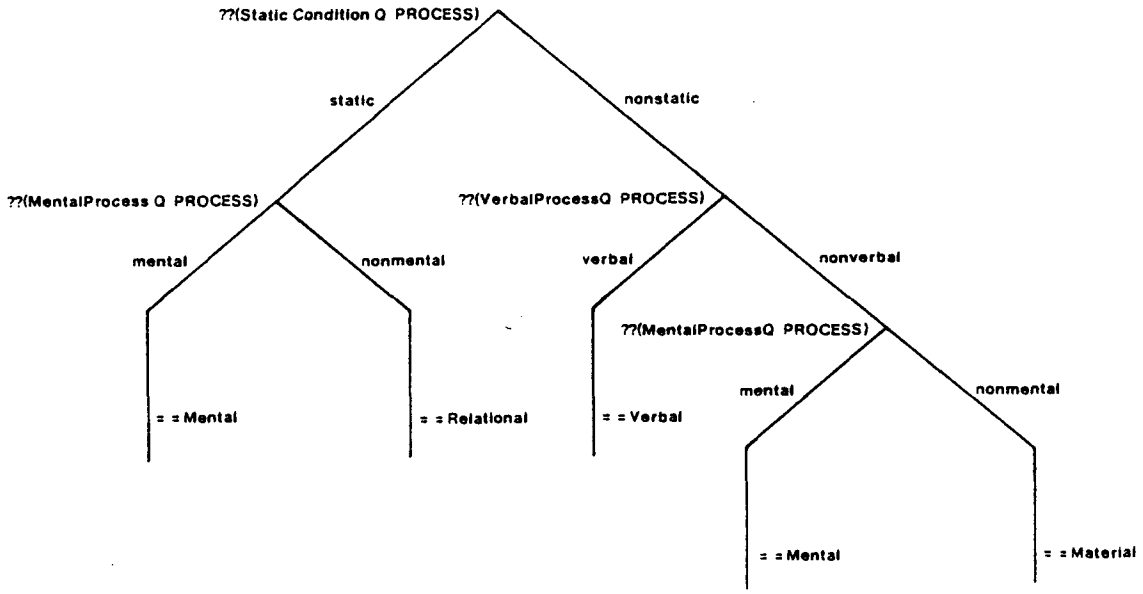

Figure 1: The Chooser of the ProcessType system

The grammar performs the final steps in the generation process. It must complete the surface form of the text, but there is a great deal of preparation necessary before it is appropriate for the grammar to start its work. Penman's design calls for many kinds of activities under the umbrella of "text planning" to provide the necessary support. Work on Nigel is proceeding in parallel with other work intended to create text planning processes.

3 The Knowledge Representation of the Environment

Nigel does not presume that any particular form of knowledge representation prevails in the environment. The conceptual content of the environment is represented in the Function Association Table only by single, arbitrary, undecomposable symbols, received from the environment; the interface is designed so that environmentally structured responses do not occur. There is thus no way for Nigel to tell whether the environment's representation is, for example, a form of predicate calculus or a frame-based notation.

Instead, the environment must be able to respond to inquiries, which requires that the inquiry operators be implemented. It must be able to answer inquiries about multiplicity, gender, time, and so forth, by whatever means are appropriate to the actual environment.

As a result, Nigel is largely independent of the environment's notation. It does not need to know how to search, and so it is insulated from changes in representation. We expect that Nigel will be transferable from one application to another with relatively little change, and will not embody covert knowledge about particular representation techniques.

4 Nigel's Syntactic Diversity

This section provides a set of samples of Nigel's syntactic diversity: all of the sentence and clause structures in the Abstract of this paper are within Nigel's syntactic scope.

Following a frequent practice in systemic linguistics (introduced by Halliday), the grammar provides for three relatively independent kinds of specification of each syntactic unit: the *Ideational* or logical content, the *Interpersonal* content (attitudes and relations between the speaker and the unit generated) and the *Textual* content. Provisions for textual control are well elaborated, and so contribute significantly to Nigel's ability to control the flow of the reader's attention and fit sentences into larger units of text.

5 Uses for Nigel

The activity of defining Nigel, especially its semantic parts, is productive in its own right, since it creates interesting descriptions and proposals about the nature of English and the meaning of syntactic alternatives, as well as new notational devices.[11] But given Nigel as a program, containing a full complement of choosers, inquiry operators and related entities, new possibilities for investigation also arise.

Nigel provides the first substantial opportunity to test systemic grammars to find out whether they produce unintended combinations of functions, structures or uses of lexical items. Similarly, it can test for contradictions. Again, Nigel provides the first substantial opportunity for such a test. And such a test is necessary, since there appears to be a natural tendency to write grammars with excessive homogeneity, not allowing for possible exception cases. A systemic functional account can also be

[11] It is our intention eventually to make Nigel available for teaching, research, development and computational application.

tested in Nigel by attempting to replicate particular natural texts--a very revealing kind of experimentation. Since Nigel provides a consistent notation and has been tested extensively, it also has some advantages for educational and linguistic research uses.

On another scale, the whole project can be regarded as a single experiment, a test of the functionalism of the systemic framework, and of its identification of the functions of English.

In artificial intelligence, there is a need for priorities and guidance in the design of new knowledge representation notations. The inquiry operators of Nigel are a particularly interesting proposal as a set of distinctions already embodied in a mature, evolved knowledge notation, English, and encodable in other knowledge notations as well. To take just a few examples among many, the inquiry operators suggest that a notation for knowledge should be able to represent objects and actions, and should be able to distinguish between definite existence, hypothetical existence, conjectural existence and non-existence of actions. These are presently rather high expectations for artificial intelligence knowledge representations.

6 Summary

As part of an effort to define a text generation process, a programmed systemic grammar called Nigel has been created. Systemic notation, a grammar of English, a semantic notation which extends systemic notation, and a semantics for English are all included as distinct parts of Nigel. When Nigel has been completed it will be useful as a research tool in artificial intelligence and linguistics, and as a component in systems which generate text.

References

[Berry 75] Berry, M., *Introduction to Systemic Linguistics: Structures and Systems*, B. T. Batsford, Ltd., London, 1975.

[Berry 77] Berry, M., *Introduction to Systemic Linguistics: Levels and Links*, B. T. Batsford, Ltd., London, 1977.

[Davey 79] Davey, A., *Discourse Production*, Edinburgh University Press, Edinburgh, 1979.

[de Joia 80] de Joia, A., and A. Stenton, *Terms in Systemic Linguistics*, Batsford Academic and Educational, Ltd., London, 1980.

[Fawcett 80] Fawcett, R. P., *Exeter Linguistic Studies*, Volume 3: *Cognitive Linguistics and Social Interaction*, Julius Groos Verlag Heidelberg and Exeter University, 1980.

[Halliday 76a] Halliday, M. A. K., and R. Hasan, *Cohesion in English*, Longman, London, 1976. English Language Series. Title No. 9.

[Halliday 76b] Halliday, M. A. K., *System and Function in Language*, Oxford University Press, London, 1976.

[Halliday 81] Halliday, M.A.K., and J. R. Martin (eds.), *Readings in Systemic Linguistics*, Batsford, London, 1981.

[Hudson 76] Hudson, R. A., *Arguments for a Non-Transformational Grammar*, University of Chicago Press, Chicago, 1976.

[Mann 82a] Mann, W. C., et. al., "Text Generation," *American Journal of Computational Linguistics* 8 , (2), April-June 1982 , 62-69 .

[Mann 82b] Mann, W. C., *The Anatomy of a Systemic Choice*, USC/Information Sciences Institute, Marina del Rey, CA, RR-82-104, October 1982.

[Mann 83] Mann, W. C., and C. M. I. M. Matthiessen, "A demonstration of the Nigel text generation computer program," in *Nigel: A Systemic Grammar for Text Generation*, USC/Information Sciences Institute, RR-83-105, February 1983. This paper will also appear in a forthcoming volume of the *Advances in Discourse Processes Series*, R. Freedle (ed.): *Systemic Perspectives on Discourse: Selected Theoretical Papers from the 9th International Systemic Workshop* to be published by Ablex.

[McDonald 80] McDonald, D. D., *Natural Language Production as a Process of Decision-Making Under Constraints*, Ph.D. thesis, Massachusetts Institute of Technology, Dept. of Electricial Engineering and Computer Science, 1980. To appear as a technical report from the MIT Artificial Intelligence Laboratory.

[McKeown 82] McKeown, K.R., *Generating Natural Language Text in Response to Questions about Database Structure*, Ph.D. thesis, University of Pennsylvania, 1982.

[Winograd 72] Winograd, T., *Understanding Natural Language*, Academic Press, Edinburgh, 1972.