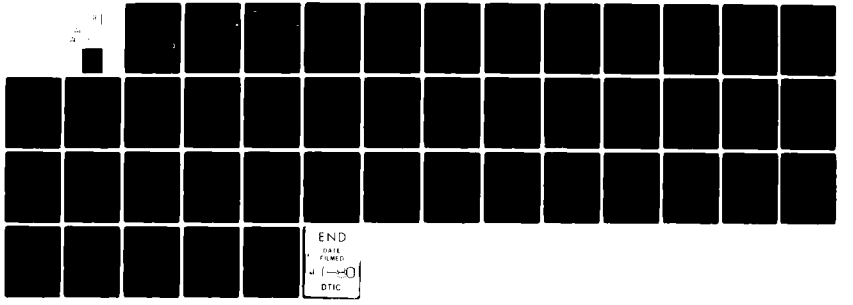
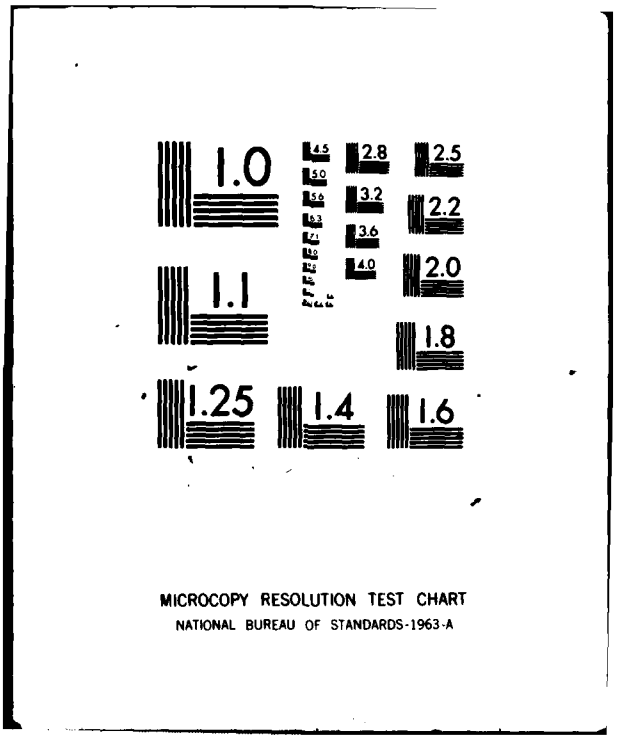
AD-A089 711 DELAWARE UNIV NEWARK DEPT OF COMPUTER AND INFORMATI--ETC F/G 5/8
PREREQUISITES TO DERIVING FORMAL SPECIFICATIONS FROM NATURAL LA--ETC(U)
AUG 80 R M WEISCHEDEL F49620-79-C-0131
UNCLASSIFIED AFOSR-TR-80-0867 NL



END
DATE FILMED
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AFOSR-TR- 80-0867

LEVEL II

(12) P.S.

Prerequisites to Deriving Formal Specifications from Natural Language Requirements: Final Report*

by

Ralph M. Weischedel

August, 1980

*Research sponsored by the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under contract no. F49620-79-C-0131. The United States Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation herein.

been specified both in English and a formal language. Portions of one, the specification of KSOS, have been studied under this contract to determine which problems are most formidable for a natural language understanding system. The most difficult problems identified by this report are ambiguity in modifier placement, nominal compounds, quantification, definite reference, and the inference of unstated relationships. Problems that prove not to be as significant are lexical gaps, lexical ambiguity, conjunction, and parenthetical expressions. The work has also identified patterns of English expression in software specification and four areas for further study.

Several practical suggestions for better documentation and for more understandable formal specifications are covered at length in a separate report. A sampling of that document is presented here.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: AFOSR-TR-80-0867	2. GOVT ACCESSION NO.: AD-A069 711
	3. RECIPIENT'S CATALOG NUMBER:
4. TITLE (and Subtitle): PREREQUISITES TO DERIVING FORMAL SPECIFICATIONS FROM NATURAL LANGUAGE REQUIREMENTS	5. TYPE OF REPORT & PERIOD COVERED: FINAL rept.
	6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): Ralph M. Weischedel	8. CONTRACT OR GRANT NUMBER(s): F49620-79-C-0131
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Dept. of Computer & Information Sciences, University of Delaware, Newark, DE 19711	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61102F 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS: Air Force Office of Scientific Research/NM, Bolling AFB, Washington, DC 20332	12. REPORT DATE: August 1980 13. NUMBER OF PAGES: 43
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

formal specifications, English specifications, modules, software design, natural language processing, software engineering

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Since English specifications and formal specifications of modules are complementary and since formal specifications require so much effort to write, this report investigates in a preliminary study whether it is feasible in the foreseeable future to have software tools which under user guidance would aid in developing precise, rigorous specifications from English ones and in detecting ambiguity and vagueness in English specifications.

This report identifies four substantial sources of modules which have

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

411182

Table of Contents

1. Purpose and Organization

For the past twelve months, a small, preliminary study, funded under Contract No. F49620-79-C-0131, has been underway on the problems a semi-automatic tool would have in understanding English specifications. This is the final technical report of that contract; this work will be extended under grant no AFOSR-80-0190.

Section 2 reviews the motivation of this study. Section 3 summarizes our work thus far. Section 4 gives further detail on our findings and tentative conclusions. Comparison of this work with that of others appears in section 5. The conclusion of the report is section 6.

2. Motivation of the Work

The project is concerned with specifications for software modules. A software module is a collection of "closely related" procedures to carry out a common task. English and formal languages are alternatives used to specify the interface of a module independent of its implementation. From the comparison in Table 1, it is clear that both types are complementary and therefore that both are important to design of large software systems. An aspect not obvious in the table is that English specifications will be the more prevalent of the two for the foreseeable future. The reason is the great difficulty (and therefore cost) of writing formal specifications (Parnas, 1976). Instances where the benefits of the formal specifications outweigh the

massive effort in writing them include:

1) developing a family of software systems (Parnas, 1976),

2) developing an embedded computer system, such as is common in military applications (Parnas, 1977), and

3) proving the correctness of aspects of the software system (Ford Aerospace, 1978).

Table 1

Characteristics of the Specification Languages

English	Formal
Easy to understand	Difficult to understand and write
Provide intuitive notion	Provide complete detail
Include both functional & nonfunctional requirements	Include only functional requirements (so far)
Frequently ambiguous	Unambiguous
Frequently imprecise	Precise
Frequently vague	Rigorous

2.1 A Machine Aid

Given the analysis above of the two kinds of languages, one naturally considers the potential of machine aids for the following tasks:

1) to detect ambiguity and vagueness in English specifications,

2) to generate the formal specifications given the English ones, and

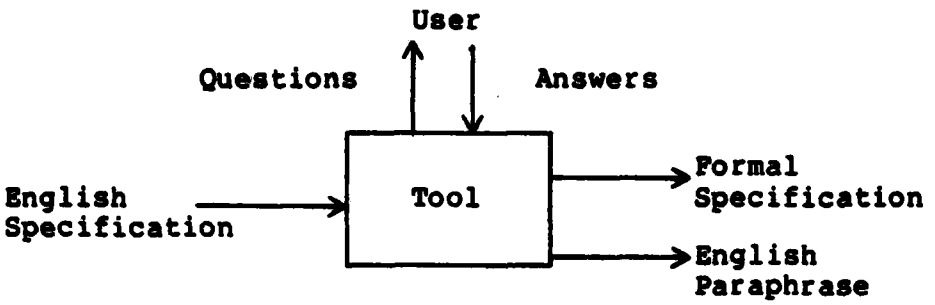
3) to verify and validate that an English

specification and a formal one define the same module interface.

The motivation of our work is that a tool of the form in figure 1 would aid in all three goals mentioned above and that several small preliminary studies would determine whether such an aid is technologically feasible in the foreseeable future. The tool would take a module's specification in English as input. Under the interactive guidance of the user (who is a software designer), the tool would generate a formal specification corresponding to the module. Guidance by the user would be supplied in direct response to questions (in English) posed by the tool regarding ambiguity, vagueness, and other aspects that the tool does not understand. An English paraphrase of the tool's understanding is given as a final output in addition to the formal specification so that the designer can easily check that the formal specification corresponds to the English input by comparing the two English versions. The English paraphrase also provides easy checking that the tool correctly understood the intent of the specification's authors.

Figure 1

Machine Aid in Specification



2.2 Hypotheses Motivating the Research

Two hypotheses together suggest that a few small studies can shed considerable insight into the technological feasibility of such a tool. The first hypothesis is that the nature of the domain may offer simplifying heuristics for understanding the English specification. For instance, the greatest success in understanding English input has been in data base retrieval, as evidenced by Artificial Intelligence Corporation marketing ROBOT, a natural language query system (Harris, 1977). Heuristics often show up as special patterns of language which relate syntax to semantics in the domain.

Second, the division of responsibility between the tool and the user (a software designer) could greatly simplify the problems in creating the tool. If the majority of tedious decisions can be made automatically

by a tool, while the most difficult for machine understanding are presented to the user for guidance, then the user interaction would greatly simplify the task of the tool without overwhelming the person with its questions. Therefore, we have been studying the language and understanding problems involved, analyzing their individual frequencies of occurrence and the success rate of present heuristics for dealing with them. The intent is not to perform rigorous statistical analysis, but merely to get a feel for the magnitude of the various language processing problems and to identify which decisions could reasonably be solved automatically by the system and which should be given to the user for guidance.

These overall goals lead us to the statement of work in the next section.

3. Summary of Completed Work

The statement of work of contract F49620-79-C-0131 is,

"a. Collect appropriate examples of computer software requirements (B-5 Spec. for example).

b. Identify frequently used English constructs and their formal language equivalents.

c. Make suggestions for improved software documentation."

Our progress is presented in sections 3.1, 3.2, and 3.3 corresponding directly to the three parts of the statement of work.

3.1 Specifications Collected

Four sources have been found providing numerous examples of both English specifications and formal specifications of the same modules. Such examples are crucial to performing part (b) of the statement of work. One is the B-5 specifications of KSOS, the Department of Defense Kernelized Secure Operating System (Ford Aerospace, 1978). Its formal specification is a 72 page, single-spaced listing written in SPECIAL (Roubine and Robinson, 1976); its English specification is 62 single-spaced pages.

A second is PSOS, a Provably Secure Operating System (Neumann, et.al., 1977). Its formal specification is 100 pages written in SPECIAL. Its English description is only 64 pages and contains less detail than a brief user's manual would. PSOS and KSOS appear to be the largest, most extensive systems ever formally specified.

A third defines the functional capabilities for a proposed secure relational data base management system (Neumann, et.al., 1977). The formal specification is 12 pages; the English one is 14 pages. Only the functional capabilities are defined, a command language interface oriented to naive users is not given.

A fourth class of examples is a textbook (Horowitz and Sahni, 1976). This text contains eight short specifications of modules that manage data structures (e.g. stacks, strings, and symbol tables). The English

specification in each case is at most a very few pages long; the formal counterpart is at most a half page in each case, written in AFFIRM (Guttag, et.al., 1978).

It will become clear in section 4 that these examples alone are more than can be analyzed in many man-years of effort. Naturally, carefully selected portions have been studied; for this contract, the portions studied were taken from KSOS.

These four samples form a good base for future analyses. Since the examples employ two widely differing kinds of formal languages the effect of the syntax or semantics of any given specification language will be factored out of future studies. The wide range in the scope of the software specified will factor out the effect of necessarily studying small portions of specifications. The small examples from the Horowitz and Sahni text will be reflective of what might be possible in the foreseeable future, while an operating system example will demonstrate the effect of a much larger semantic domain.

3.2 English Constructions and Formal Language Counterparts

We have intensely analyzed 110 sentences from three portions of the KSOS specification (Ford Aerospace, 1978); each of the three parts was distinctly different in purpose. One of our goals in this intense analysis has been to find frequently used English constructions that might be incorporated into a machine aid for

generating formal specifications; such patterns offer simplifying heuristics for the tool. The second goal has been to isolate problems that the machine can resolve automatically and those that should be passed back to the designer; results regarding the second goal are presented in section 4.

Table 2 summarizes the patterns we have found and the meaning that the tool would assign to them.

In general, we have not counted as patterns the special syntactic and semantic constraints imposed by main verbs in a clause. These are patterns that are used directly as heuristics in limited domains of discourse to reduce ambiguity. In some styles of writing parsers, such as semantic grammars (Burton, 1976) they would likely be encoded as patterns. In another style, where semantics is a separate component from syntax but nonetheless directs the parser through frequent interaction, these constraints would appear in the dictionary, not as explicit patterns in the parser. We intend to use the latter style of parser, and have not included such constraints in our table of patterns. Those constraints should prove very effective heuristics however, for they do significantly limit the cases of syntactic ambiguity in the KSOS portions studied and there is very little lexical ambiguity present. (Lexical ambiguity refers to a word having more than one possible sense). This is covered further in section 4.

Table 2

Some English Patterns in Software Specifications

(For expository purposes, we present simplified patterns in BNF, though the patterns would fit into many different grammatical styles. In addition, many syntactic paraphrases of these patterns would be included in a realistic grammar. Nonterminal symbols are enclosed in angle brackets. Other symbols are terminals except for the distinguished symbol '::=' and the angle brackets. Parentheses are also terminal symbols. All page references are to Ford Aerospace (1978).)

1. <noun phrase> ::= <noun phrase> (<acronym>)

   This is a common way of defining an acronym, e.g. "Kernelized Secure Operating System (KSOS)". The pattern should also have the constraint that the letters of <acronym> are all capitalized and are the first few letters of the last few words of <noun phrase>.

2. <noun phrase> ::= <noun phrase>$_1$ of <noun phrase>$_2$

   where <noun phrase>$_1$ must refer to a data structure and <noun phrase>$_2$ must refer to data elements for the pattern to be matched. This is used to indicate the type of elements in a data structure, e.g. "a set of integers," "a sequence of characters," or "a data base of name space partitions in use."

3. <head noun> ::= <head noun> [<name> <year>]
   <sentence> ::= <sentence> [<name> <year>]

   "Head noun" is the central noun of a noun phrase; words before and after it are either premodifiers or postmodifiers of the head noun. Both of the patterns are a way of giving a bibliographic reference for a concept defined either by the noun phrase containing the head noun of the first pattern or by the sentence in the second pattern. An example of the first is, "The Security Kernel shall be used to support the KSOS UNIX Emulator [Emulator 78] which..." (p. 8). An example of the second is, "Integrity is defined as the formal mathematical dual of security [Biba 75]" (p. 2). A <name> itself has several forms: <surname>, <surname> and <surname>, <surname> et al., <noun>, <acronym>, <abbreviated noun>. The first three of course identify the reference by author name, whereas the last three do so by a keyword of the title.

4. <noun phrase> ::= <premodifiers> instance of <noun phrase>$_1$

<noun phrase> ::= <premodifiers> example of <noun phrase>$_1$

This is a means of describing or defining members of a set explicitly or implicitly given by <noun phrase>$_1$. An example where the set is implicit is given by, "The name within the partition shall be used to specify particular instances of an object within the partition" (p. 9).

5. <sentence> ::= <noun phrase>$_1$ <be> returned as the result of <noun phrase>$_2$

<noun phrase>$_2$ must refer to a function, activity, or process of the software module. <be> matches any form of the word "be". This pattern is used to define a particular value, or more generally the class of values returned by a function or process of the software module. It tells more than the data type of the result, giving more detailed semantics for the value returned and therefore more detailed semantics for the function. An example is "A SEID shall be returned as the result of new object creations (i.e. K_create, K_build_segment,...)" (p. 9).

6. <noun phrase> ::= <noun phrase>$_1$ called <noun phrase>$_2$

This indicates that the item referred to by <noun phrase>$_1$ is named <noun phrase>$_2$. An example is "All Kernel objects shall be referenced by a common designator called The Secure Entity Identifier..." (p. 9).

7. <noun phrase> ::= <noun phrase>$_1$: <noun phrase list>

where <noun phrase>$_1$ is plural and denotes a set and where <noun phrase list> is a list of conjoined noun phrases. The system upon seeing this can infer that the items listed in <noun phrase list> are elements of the set given by <noun phrase>$_1$ and have the same data type as <noun phrase>$_1$.

8. <adjunct> ::= regardless of <noun>

where <noun> is the name of a class of entities. An <adjunct> may appear at many places in a sentence and modifies the sentence as a whole. This can be approximated in software specifications as "for all <noun>s" and therefore corresponds to the universal quantifier. An example is "All process segments, regardless of domain, shall be manipulated by the same set of Kernel segment primitives" (p. 12).

9. <identifier definition> ::= <identifier> : <noun phrase>

   where <noun phrase> may be very cryptic, requiring special semantic processing. This is a means used throughout the document to define the purpose of an identifier, either as an argument to a procedure or as a value returned. It occurs after a stereotyped form, such as "The Kernel primitive... shall take the following parameters". An example is, "vAddr : virtual address of first byte in the segment" (p. 21).

10. <noun phrase> ::= <premodifiers> supply of <noun phrase>$_1$

    where <premodifiers> are the normal words that can appear before the main noun of a noun phrase and <noun phrase>$_1$ must be a plural or collective referring to some resource. In the software environment, this can be treated as a description of a set of entities, whose members all satisfy the description of <noun phrase>$_1$. An example is, "the supply of mapping registers" (p. 21).

11. <sentence> ::= <noun phrase>$_1$ initialize <noun phrase>$_2$to <noun phrase>$_3$

    where <noun phrase>$_1$ must be a process or person, <noun phrase>$_2$ must refer to memory, and <noun phrase>$_3$ must denote a value or state. Of course, this defines initial values for some unit of memory. An example of a passive sentence corresponding to this form is "The segment shall be initialized to contain all zeros" (p. 21).

12. <sentence> ::= <noun phrase>$_1$<be> shared between <noun phrase>$_2$

    where <noun phrase>$_1$ is some resource and <noun phrase>$_2$ must be plural and must be a resource user. Naturally, this is a description of resource sharing. An example is, "Segments may be shared between processes..." (p. 21).

3.3 Suggestions for Improved Software Documentation

Documentation is a rather all encompassing term, including the English specifications, comments and descriptions of the formal specification, and the formal specification itself. Our suggestions in this study

range over those three possibilities also.

In addition to the 110 sentences intensely analyzed we have performed some less intense analysis of larger sections of the KSOS specifications to see the context of the portions analyzed intensely and to add more data on which to base our suggestions regarding documentation. In the less intense analysis, we have primarily looked for three things:

1) striking features of the English descriptions,

2) the corresponding portion of the formal specification, and

3) items that cause the investigator to reread.

The items that cause the investigator to reread a sentence or section before feeling that the meaning was understood are oftentimes signals of poor documentation.

Suggestions for improved documentation must be understandable to the normal designer, technical writer, and programmer, who write them; hence, they cannot be couched in terms of artificial intelligence or computational linguistics.

Our practical suggestions for improved documentation are discussed at length in Weischedel (1980). Also hypothesized there is a list of causes for formal specifications being so difficult to understand and write. A sampling of the suggestions in Weischedel (1980) are presented next; see the report for further information.

1) <u>English specifications can and should provide</u>

summaries of detail cutting across several functions of a module's interface. Formal specifications thus far tend to present detail for each function in an isolated way, making them harder to understand.

2) English specifications can and should state explicitly important implications of a formal specification which are stated only implicitly in the formal version.

3) A cross-reference between the English specification and formal specification is invaluable both for understanding the formal specification and for informally validating that it does what the English description claims.

4) For long specifications, an index is necessary. For lack of a better measure, we arbitrarily define "long" for formal specifications as at least six pages; "long" for English specifications we arbitrarily define to be at least a dozen pages.

5) Certain forms in English descriptions are easily abused, leading to incomplete understanding. Parentheses should only be used for short renamings of an entity, such as "the Kernelized Secure Operating System (KSOS)" or "k_build_segment (a kernel primitive)". A cryptic, telegraphic style should never be used, not even to define the meaning of variables. In both of these cases, a clearer paraphrase can usually be found without using the misused form.

4. Additional Findings

For each sentence analyzed from the KSOS specification, the following items were identified:

1) a semantic representation as a typical parser might generate,

2) any corresponding features in the formal specification,

3) any problems for machine understanding, and the demands those problems place upon a system's dictionary, parser, semantic system, world knowledge, and reasoning,

4) the applicable heuristics to those problems, and

5) the role the sentence plays in the whole.

Identifying these aspects not only enables us to find the kind of valid patterns sought, but also provides data for factoring the problem-solving into that which the machine can do automatically and that which should be posed to the user.

Such painstaking analysis is quite time-consuming empirical observation. There are two major reasons for this: There simply are no detailed empirical studies in the literature of the style of English used in software specifications; so, this is uncharted territory. Second, understanding sentences occurs for the most part below the conscious level of reasoning; consequently, introspection and intuition can offer very little aid. Because one is not conscious of alternate interpretations, very careful thought and observation is required to hypothesize the various interpretations and to

ascertain which problems are solvable by present heuristics.

Such detailed analysis averages roughly two hours per sentence for an initial study; review by a second individual and recording the conclusions takes roughly another hour per sentence. At first this sounds rather horrendous. However, the only alternative is to build a prototype of the tool and empirically analyze it while processing sentences. To analyze several hundred sentences our way requires several man-months therefore. From our experience in building three understanding systems (Joshi and Weischedel, 1976, Weischedel, 1979, and Weischedel, et.al., 1978), we estimate it would take several man-years of effort to build a system to analyze an equivalent number of sentences. Hence, our empirical methodology requires about an order of magnitude less effort than it would to build a large system for the same empirical results.

In section 4.1, we specifically discuss the problems for machine understanding of English specifications. Section 4.2 presents problems for generating appropriate English paraphrases of the formal specification constructed.

4.1 Difficulties for Machine Understanding

There are several specific problems we looked for. For each, we give an example, refer to relevant heuristics, and state tentative conclusions. Our conclusions at this point are qualitative rather than quantitative;

the body of data is simply too small yet to draw quantitative conclusions. Under the renewed support for this work, given through grant number AFOSR-80-0190, the body of data will be enlarged and additional heuristics will be examined.

1) Lexical gaps. The lexicon, the linguistic term for a dictionary, is a major component of a natural language understanding system. We divided the words actually occurring in the analyzed sentences into three groups: One contains words that would be expected to occur in general software environments; these, we assume, would be part of the software tool for aiding designers in creating specifications. Examples include "the", "return", and "integer". A second is comprised of words that are closely related to KSOS, rather than to software in general; these would have to be added to the lexicon to prepare the general software tool for the concepts related to KSOS and to other secure operating systems. We found this class to be substantial, but rather predictable, since it consists largely of proper names (e.g. "Kernel" and "Emulator") and security related terms (e.g. "trusted" and "discretionary").

The third class is a problem, however; it consists of terms that might not be anticipated in building the software tool. Examples are "slightly", "complex", "residence", and "exhaust". For unanticipated terms, there are two kinds of approaches one can take. Trying to infer their meaning is in general beyond the state of

the art, though some investigations have begun in this area (Carbonell, 1979; Goldman, et al., 1977; and Miller, 1975). Alternatively, one could ask the user for a paraphrase in terms it already knows or for permission to ignore the word. Hendrix, et al. (1978) uses this in a data base environment; there is insufficient evidence of how frequently this approach will work.

Even if the heuristics for dealing with new words never approach the success displayed by humans, the severity of this problem is reduced by the fact that users of natural language systems tend to tailor their input to perceived system limits (Harris, 1977 and Malhotra, 1975). Consequently, though the problem of unexpected words is one of the most scientifically challenging, it is difficult to predict whether it will ultimately prevent a software tool from being effective. We feel the flexibility of persons in using such a tool will prevent lexical gaps from being a severe handicap.

2) Conjunction. Ambiguity arising from the conjunctions "and" or "or" has been known as a difficult problem since the earliest natural language understanding systems (Woods, 1973). Roughly one quarter of the 110 sentences analyzed contained conjunctions. In only two cases did the use of conjunction seem unparsable using current techniques, such as SYSCONJ (Woods, 1973). However, the remaining cases seem to divide almost evenly between those where heuristics could reliably determine the intended meaning and those where the user would

need to be asked to disambiguate it. In many cases where the user would be asked, the alternate interpretations would be laughable to a person. Clearly, better heuristics are needed to resolve ambiguity related to conjunction automatically. However, it is usually rather easy (though verbose) for the user to restate the input without using the constructions that are most difficult to understand. For instance, "The virtual address and domain parameters shall be for the calling process only" can be restated in two sentences, "The virtual address parameter shall be for the calling process only. The domain parameter shall be for the calling process only." Alternatively, one can easily state this in a less verbose way which the system would not find ambiguous; namely, "The virtual address parameter and the domain parameter shall be for the calling process only." Since it is often easy to rephrase a use of conjunction in a way that the system would find unambiguous, and since techniques for precisely notifying users of difficulties in processing an English input are becoming available (Weischedel and Black, 1980), we feel that an effective strategy for dealing with conjunction can be worked out for the proposed software tool by combining the tool's heuristic with the person's ability to paraphrase.

3) Lexical ambiguity. Lexical ambiguity arises when a word may have more than one meaning (technically termed "sense") in a sentence. Of all the possible

problems for natural language processing in our intense analysis, this is perhaps the easiest to overlook in a given sentence. We found relatively few instances of lexical ambiguity, and they involved only a handful of words. An example is "may" which can mean "is permitted" or "is possible". Since the number of instances in the sample is small, techniques such as those suggested in Small (1979) should prove effective when adapted to a standard parser using semantic interaction.

4) Modifier placement. Certain combinations of syntactic constructions in English are ambiguous because it is unclear what one of the phrases modifies. Usually (but not always) a person can resolve the ambiguity in context via semantic and pragmatic factors, but techniques for automatic resolution of the ambiguity by machine are an open area of research. In "K_invoke can remove any special privileges for the process", present systems would have difficulty determining whether the privileges are being removed on behalf of the process or whether the privileges of the process are being removed. However, in context it is not ambiguous to a person reading it. Roughly one quarter of the sentences exhibited modifier placement ambiguity. There are two independent heuristics for this: One has been implemented in the RUS parser (Bobrow and Webber, 1980), and tries to capitalize on semantic information to direct the parse. The other (Heidorn, 1976) tries to capture an apparent heuristic about the way people organize

sentences with such modifiers. In the follow-on grant to this contract, we will be examining the effectiveness of these two heuristics not only on these 110 sentences but also in a prototype system using the RUS parser.

5) Other syntactic ambiguity. Ambiguity related to syntax, other than the ambiguity mentioned in items (2), (3), and (4), occurred in only ten of the sentences analyzed. The heuristic, mentioned in (4), which uses semantic information to direct the parse would leave ambiguity in only seven of the instances. Since the cases of ambiguity arise so infrequently, this should not be a problem for processing English specifications semi-automatically, as long as good techniques become available for posing the alternatives, paraphrased in English, to the user. Techniques for such paraphrasing will be studied in the follow-on grant.

6) Nominal Compounds. Nominal compounds are strings of nouns or nominalized verbs; the meaning of the string is not just the sum of the meanings of the individual words. For instance, "new object" can be considered as an object which is also new; however, the nominal compound "user mode" is not a mode which is also a user. Determining the meaning of nominal compounds is therefore a problem for mechanical processing. Furthermore, if the string consists of more than two words, there is ambiguity. For instance, in "user mode programs" the grouping is leftward; one is talking about programs running in the user mode, not about a kind of

mode program associated with users. On the other hand, the grouping can be to the right, as in "KSOS UNIX Emulator", which is the UNIX Emulator associated with KSOS, as opposed to an Emulator for KSOS UNIX. Nominal compounds occurred in roughly 70% of the sentences; strings consisting of three or more words were not uncommon. Finin (in press) deals with the problems of computational models of understanding nominal compounds. Also, with a semantically directed parser, such as Bobrow and Webber (1980) report, it may be possible to implement the suggestions in Brachman (1978) for understanding nominal compounds. However, these strategies were too recent to consider during the contract period. Investigating a knowledge representation language which could support the understanding of nominal compounds is an emphasis of the follow-on grant.

7) Quantification. The scope of quantifiers poses an ambiguity that has interested philosophers, linguists, and computer scientists for many years. For instance, "Only one object shall be assigned to each partition", could mean that the same object (and only it) is assigned, or that each partition could have a different object assigned to it (though only one in each case). Woods (1978) has a heuristic for determining quantifier scope, but Van Lehn (1978) argues that a completely new approach is needed. Quantifier scope is just one aspect of a much larger problem: determining what if any logical quantifiers should be used to

represent a given noun phrase. Plural noun phrases are sometimes properly represented by the universal quantifier to indicate that a fact or activity is true for each individual in a set, but other times they represent a collective set of entities acting together rather than as individuals. Indefinite noun phrases, such as "a process", sometimes correspond to some particular entity, which would be represented by the existential quantifier; at other times they mean a generic example or norm for a class of entities rather than an individual. There has been very little discussed regarding computational techniques to solve these two problems of interpretation; yet in our sample, these two classes of decisions arose more frequently than the classical decision regarding quantifier scope.

Van Lehn (1978) and Bobrow and Webber (1980) correctly suggest that a new computational view of quantification is needed. We will be studying these problems from another perspective in the follow-on grant. From the mechanisms of conveying quantified information in English, we hope to suggest alternatives for specifying that information in formal languages. Psychological evidence (Thomas, 1976 and several references therein) has shown that people not trained in mathematical logic have great difficulty in interpreting quantification in English. Therefore, alternatives to logical quantification might significantly aid the understandability of formal specifications.

8) Additional semantic problems. An additional semantic problem is understanding telegraphic style. For instance, in context, "Rendezvous Name (SEID) already in use at or below the level of this process" clearly means, "The rendezvous name (the SEID) is already in use at or below the level of this process." Yet, the missing relationships represented by the underlined words must be inferred. Work on this significant problem is just beginning. Two different approaches to the syntactic aspects of the problem have been presented by Kwasny and Sondheimer (1979) and Hayes and Reddy (1979). However, the semantic problem of inferring the missing relationships has not been attempted in a general way.

Another class of problems that arise frequently in our sample is inferring the relationship of parenthesized material to non-parenthetical. The example of cryptic style above demonstrates a case where the syntax makes the relationship clear. The parenthetical material "SEID" is an appositive giving a proper name for the noun phrase "rendezvous name". Roughly one twelfth of the sentences studied contained parenthetical material whose relationship to the nonparenthetical would be very difficult for a machine to infer. No heuristics for this problem are available yet. Even if a heuristic for this is never available, the system could request the user to restate difficult cases of parenthetical material. This has an additional advantage; oftentimes

restating the problematic examples of the sections studied yields clearer English for the purpose of understanding by humans.

Of course, the semantics of a number of words used is not yet represented well computationally. However, there is no significant trend evident here. An example of a problematic case is the word "somewhat". Since no trend emerged, meanings for necessary words could be developed on a case by case basis.

One additional problem is inferring the way the individual sentence fits into the whole. This has been called variously "intersentential relations," "coherence relations," or "structure" in the document. The structure of a paragraph is very difficult to detect, as an example from process descriptions in Balzer, et.al. (1978) shows. Even where structure is explicitly indicated using deeply refined subheadings such as "3.1.1.2.1.1 Kernel Object: Processes" or is explicitly indicated using indentation, no techniques have been developed to detect and use such explicit clues.

9) Definite Reference. Closely related to the problem of inferring conceptual structure relating a sentence to the whole is the effect of structure on what a pronoun or noun phrase refers to. An example of this impact is the first sentence of Section 3.1.1.2.1.1 "Kernel Object: Processes," which is "In KSOS the process is the only active agent in the system." "The process" in the context immediately after the title does

not refer to a particular process as would normally be the case, but rather refers to the generic notion of processes in the KSOS Kernel. Noun phrases or pronouns which would be processed by heuristics for definite reference occurred in roughly two-thirds of the sample sentences; many of the sentences contained more than one instance of the phenomenon. Grosz (1977), Hobbs (1979), Sidner (1979), and Webber (1978) present heuristics for resolving and understanding references, given that the text's structure is known or detectable. It is not clear in our analysis thus far how effective the heuristics will be for the frequent use of pronominal and noun phrase reference in software specifications. However, the problem of inferring the structure relating a sentence to the whole remains. We speculate that since the possible discourse structure appears more complex in software specifications than it does in present data base query systems, the simple heuristics that work adequately in that domain are not likely to be rich enough for software specifications. More analysis of the severity of the problem and of the adequacy of the heuristics in the papers above is needed.

Since the nine classes of problems raised above will not always be resolved without user guidance, the system must be able to generate questions to obtain user guidance at particular points of difficulty. Naturally, this requires that the questions be stated in English

and that the system have a clear understanding of what the alternatives are. The need for such clarifying questions has been discussed by Codd, et.al. (1978) and Hayes and Reddy (1979). The approach proposed by Hayes and Reddy (1979) would limit their heuristics to simple domains; the domain of software specifications is far too rich to be a simple domain. McKeown (1979) has shown that much care is needed in the system's choice of phrasing, and has offered heuristics for one aspect of the general problem of choosing a correct phrasing. Interaction for eliciting user guidance is an area needing further development.

4.2 Problems in Generating English Paraphrases

There are at least two aspects to generating the English paraphrase of the formal specification which require an intermediate level of knowledge representation. One is that properties implicit in the formal specification oftentimes should be explicitly stated in an English paraphrase. For instance, in KSOS the functions K_create, k_build_segment, k_create_device, k_fork, and k_spawn in their formal specifications each return a value whose type is seid. This is given in five separate statements, one for each of the functions. Yet, this implicit fact is quite usefully summarized in the English specification by "A SEID shall be returned as the result of new object creations (i.e. k_create, k_build_segment, k_create_device, k_fork, and k_spawn)." An intermediate level knowledge representation would

summarize such facts formally.

The second aspect, which happens to be illustrated by the same example, is that the organization of information should be different. What is implicit in the formal specification is often stated explicitly in English specifications. Ideally, the intermediate knowledge representation by its organization of information would relate the English input to appropriate parts of the formal specification as output. Then, the intermediate knowledge representation would serve as the basis for an English paraphrase which would likewise relate the input to the appropriate parts of the formal specification.

Mann and Moore (1979) discuss computer generation of English texts from a knowledge base. The kind of knowledge base their future work will consider is process descriptions written in AP2 (Balzer, in preparation). The work we suggest is complementary to theirs, namely, studying the kind of knowledge representation and structure that describes modules (rather than just processes) and which would be the input to an English text generation system.

Another kind of English output needed is answers to questions posed by the designer. The designer must be able to ask the system why certain decisions were made. For, it is likely that the designer will want to check the rationale for some of the decisions in order to be comfortable with the choices made.

An appropriate knowledge representation is central to several of the problems determining the technical feasibility of the proposed machine aid. In addition to the ones in this section regarding generating English output, it is also crucial to the following problems in understanding the English input: representing the meaning of nominal compounds and inferring missing relationships.

5. Related Work

In this section, work which is related to our project in overall goals is presented. The comparison leads us to conclude that our project is unique in complementing related projects.

Mander and Presland (1979) describes a program for specification analysis (SPAN) which they are developing. Their goal is to process English requirements documents a user might write, yielding two kinds of output: 1) notification of places of potential ambiguity in English and 2) suggested lists of objects and actions as a starting point for a user to write the requirements in a notation such as SA (Ross, 1977) or PSL/PSA (Teichroew and Hershey, 1977). Though their goals are well justified, the mechanisms they are implementing in SPAN seem inadequate to the task. SPAN will analyze the requirements document with a purely syntactic analysis and will note some places of possible ambiguity, such as multiple modifier placement and conjunction. Unfortunately, in our analysis of module specification, we have found that

the number of those kinds of places which are genuinely ambiguous to a person is rather small. Thus, we conclude that their analyzer will flood the user with projected ambiguities which do not exist, since semantics and intersentential context is not modeled in SPAN. Similarly, other sources of ambiguity are not detected, such as references via pronouns and definite noun phrases. (Following the strategies they use in the other cases, one could flag every pronoun and definite noun phrase, but this would also flood the user with cases that are not ambiguous in context.) Since no understanding is employed in SPAN, it does generate candidates for actions and objects which are not reasonable. In summary, though the goals are worth pursuing, an approach based on processing programming languages is inadequate for those goals.

Hobbs (1977) has made some observations on the differences between programming languages and procedures described in English. However, the observations are very general rather than concrete and are for programming languages rather than module specification languages.

Miller (1977) reports on an in-depth linguistic study of roughly 60 kitchen recipes published in a popular cookbook; the long-term aim of this project at IBM's Watson Research Center was a step toward programming in English. They specifically looked for linguistic mechanisms used to describe processes, but their choice

of domain was inappropriate since kitchen recipes are only obliquely related to their stated long-term goal regarding software. Since there is no formal language specification corresponding to a kitchen recipe, their study does not establish the needed link between English and a formal language nor does it isolate the problems in translating from English to formal.

Quite a number of projects have been undertaken in the area of program synthesis. These have tended to involve building a large system which takes as input a description of the desired input-output behavior and generates a program having that input-output behavior. Examples of such research include Balzer, et al., (1978), Barstow and Kant (1976), Bierman and Ballard (1980), Green (1976), Heidorn (1972), Manna and Waldinger (1977,1978) and Hammer and Ruth (1979).

Our work is completely unique when compared to all projects in program synthesis or related areas based on the following grounds:

1) The difficult issues of data structure selection in program synthesis do not arise in our work, since data structure selection is an implementation decision and since modules must be specified independent of implementation.

2) The formidable problem of algorithm selection in program synthesis does not arise either, since that too is an implementation decision.

3) There are no efficiency concerns in the

generated specification, since one does not execute a module specification in a production environment as one does a program.

4) None of the program synthesis projects have an emphasis on natural English input. Many assume the input is in a variant of first order predicate logic. The others have highly restricted English. For instance, SAFE (Balzer, et.al., 1978) takes only fully, unambiguously parsed English as input, where numerous parentheses to indicate the parse must be added by hand prior to input.

5) The emphasis of our work is on a series of small, preliminary studies of English specifications and formal ones, rather than building a large system, as a means of assessing whether the proposed tool will be feasible in the foreseeable future. In fact, as argued in section 3, the kind of small studies we have undertaken appear to require about an order of magnitude less effort than examining the same number of examples would take in developing a large system to process them.

Though this project is so different compared to program synthesis research, it contributes to it in a complementary way. One contribution is that for any significantly sized piece of software, it will need modules. Hence, synthesis of large systems will require dealing with modules. A second contribution, of course, is that an ideal form of input for program synthesis of software containing modules is an English description

rather than a formal one, for the reasons given earlier.

6. Conclusions

Clearly, generating the formal specification of KSOS (72 pages) from its English description (62 pages) is not technologically feasible in the foreseeable future. The semantic scope of the concepts discussed and the amount of contextual structure influencing understanding is far richer than in data base question answering, the domain in which natural language processing has achieved its greatest success. The major purpose in selecting KSOS for our intense analysis under this contract was to examine a realistic, complex example to get the broad picture of the depth and breadth of problems that a semi-automatic tool to process English specifications would have. Small examples, of the type that might be processed in the foreseeable future, would not give the broad picture necessary initially.

The following problems are the most difficult for such a tool processing English specifications:

i) ambiguity in modifier placement,

ii) nominal compounds,

iii) quantification,

iv) pronoun and noun phrase reference, and

v) inferring missing relationships.

This is based on the frequency with which the problems occurred in the sample, the difficulty with which users could rephrase the English to avoid the problem, and the

likely success of available heuristics. (However, we were not able to assess during the contract period whether adequate heuristics are available already for the reference problem, item (iv), due to the number of recent publications regarding it.)

On the other hand, the following problems are not as significant for such a tool:

vi) lexical gaps,

vii) lexical ambiguity,

viii) conjunction, and

ix) parenthesized expressions.

This is based on the frequency of occurrence, on the existence of heuristics to handle simple cases, and on the ease with which people could adjust to paraphrasing complex cases into processable forms.

Our results suggest four areas of future work.

1) We suggest continued analysis of English specifications and their formal language counterparts for several reasons. 1) As indicated in the proposal for this contract, the data one can collect in one year is simply too small to guarantee firmer, sharper conclusions. 2) Examining smaller specifications than KSOS will suggest how much the problems diminish as the length of the specification diminishes and the variety of modules in the domain becomes more constrained. 3) Examining samples whose formal specifiction language differs widely from the semantics of SPECIAL will factor out that influence. Additional sources for such

analysis have been found (see section 3.1) as part of the statement of work of this contract.

2) A second emphasis is the development of an intermediate-level knowledge representation. This is critical to solving both (ii) and (v) above. As indicated in section 4.2, it is also critical to paraphrasing the formal specification into an English output, and explaining the reasons behind the tool's decisions.

3) A third area of future work is developing and testing heuristics for resolving ambiguity and generating clarifying questions. This is crucial for effectively obtaining user guidance without unduly burdening the user. As much ambiguity as possible, particularly that arising from items (i) and (iii) above, must be resolved by the machine automatically.

4) An alternative to quantification in formal languages based on an analysis of the mechanisms in English for conveying quantified information is called for, since psychological experiments (Thomas, 1976 and several references therein) have shown that people not trained in mathematical logic have great difficulty in interpreting quantification in English.

We have drawn a number of conclusions as to why formal specifications are so difficult to understand. These are presented in Weischedel (1980), along with our practical suggestions for making formal specifications more understandable.

Acknowledgements

The assistance of Linda Salsburg in our study of portions of KSOS has been invaluable.

7. References

Balzer, Robert, "AP2 Reference Manual," Information Sciences Institute, Marina del Rey, CA, (in preparation).

Balzer, Robert, Neil Goldman, and David Wile, "Informality in Program Specification," IEEE Transactions on Software Engineering, Vol. SE-4, 2, March, 1978.

Barstow, David R. and Elaine Kant, "Observations on the Interaction Between Coding and Efficiency Knowledge in the PSI Program Synthesis System," 2nd International Conference on Software Engineering, IEEE Computer Society, IEEE Catalog No. 76CH1125-4C, October, 1976.

Biermann, Alan W. and Bruce W. Ballard, "Toward Natural Language Computation," American Journal of Computational Linguistics, 6, 2, 1980.

Bobrow, R. J. and B. L. Webber, "PSI-KLONE - Parsing and Semantic Interpretation in the BBN Natural Language Understanding System", In CSCSI/CSEIO Annual Conference, CSCSI/CSEIO, 1980.

Brachman, R. J., "A Structural Paradigm for Representing Knowledge," Report No. 3605, Bolt Beranek and Newman Inc., Cambridge, MA, 1978.

Burton, Richard R., "Semantic Grammar: An Engineering Technique for Constructing Natural Language Understanding Systems", BBN Report No. 3453, Bolt Beranek and Newman, Inc., Cambridge, MA, December, 1976.

Carbonell, Jaime G., "Toward a Self-Extending Parser," in Proceedings of the 17th Annual Meeting of the Association for Computational Linguistics, San Diego, August, 1979, 3-7.

Codd, E. F., R. S. Arnold, J-M. Cadiou, C. L. Chang and N. Roussopoulis, "RENDEZVOUS Version 1: An Experimental English-Language System for Casual Users of Relational Data Bases, IBM Research Report RJ 2144, San Jose, CA, January, 1978.

Finin, Timothy L., "The Semantic Interpretation of Noun-Noun Modification", Technical Report, Coordinated Science Laboratory, University of Illinois, Urbana, IL, (in press).

Ford Aerospace, "Secure Minicomputer Operating System (KSOS): Computer Program Development Specifications (Type B-5)," Tech. Report No. WDL-TR7932, Ford Aerospace & Communications Corporation, Palo Alto, CA, 1978.

Goldman, N., R. Balzer, and D. Wile, "The Inference of Domain Structure from Informal Process Descriptions",

Proceedings of the Workshop on Pattern Directed Inference Systems", SIGART Newsletter, 63, 1977, 75-82.

Green, C., "The design of the PSI program synthesis system," in 2nd International Conference on Software Engineering, Long Beach, CA: IEEE Computer Society, October, 1976.

Grosz, Barbara J., "The Representation and Use of Focus in Dialogue Understanding," Technical Note 151, SRI International, Menlo Park, CA, 1977.

Guttag, John V., Ellis Horowitz, and David R. Musser, "Abstract Data Types and Software Validation," CACM, vol. 21, number 12, December, 1978, 1048-1063.

Hammer, Michael and Gregory Ruth, "Automating the Software System Development Process," Research Directions in Software Technology, Peter Wegner (ed.), Cambridge, MA: The MIT Press, 1979.

Harris, L. R., "User Oriented Data Base Query with the ROBOT Natural Language Query System," International Journal of Man-Machine Studies, 9, 697-713, 1977.

Hayes, P. and R. Reddy, "An Anatomy of Graceful Interaction in Spoken and Written Man-Machine Communication," Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, August, 1979.

Heidorn, G. E., "Natural Language Inputs to a Simulation Programming System," Technical Report NPS-55HD72101A, Naval Postgraduate School, Monterey, CA, October, 1972.

Heidorn, George E., "An Easily Computed Metric for Ranking Alternative Parses", Presented at the 14th Annual Meeting of the Association for Computational Linguistics, 1976.

Hendrix, Gary G., Earl D. Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum, "Developing a Natural Language Interface to Complex Data," ACM Transactions on Database Systems, Vol. 3, 2, June, 1978, 105-147.

Hobbs, Jerry R., "What the Nature of Natural Language Tells Us About How to Make Natural-Language-Like Programming More Natural," SIGPLAN Notices, 12, 8, 1977, 85-93.

Hobbs, Jerry R., "Coherence and Coreference," Cognitive Science, Vol. 3, Number 1, 1979, 67-90.

Horowitz, Ellis and Sartaj Sahni, Fundamentals of Data Structures, Computer Science Press, Inc., Woodland Hills, CA, 1976.

Joshi, Aravind K. and Ralph M. Weischedel, "Some Frills for Modal Tic-tac-toe: Semantics of Predicate Complement Constructions," IEEE Transactions on Computers, C-25(4), 1976, 374-389.

Kwasny, Stan C. and Norman K. Sondheimer, "Ungrammaticality and Extragrammaticality in Natural Language Understanding Systems," in Proceedings of the 17th Annual Meeting of the Association for Computational Linguistics, San Diego, August, 1979, 59-63.

Malhotra, Ashok, "Design Criteria for a Knowledge-Based English Language System for Management: An Experimental Analysis", MAC TR 146, Projct MAC, Massachusetts Institute of Technology, Cambridge, MA, February, 1975.

Mander, K. C. and S. G. Presland, "An Introduction to Specification Anaysis - SPAN", Dept. of Computational and Statistical Science, The University of Liverpool, Liverpool, 1979.

Mann, William C. and James A. Moore, "Computer Generation of Multiparagraph English Text," USC/Information Sciences Institute, Marina del Rey, CA, 1979.

Manna, Zohar and Richard Waldinger, "The Automatic Synthesis of Recursive Programs," Proceedings of the Symposium on Artificial Intelligence and Programming Languages, SIGPLAN Notices, 12, 8, 1977, 29-36.

Manna, Z. and R. Waldinger, "The Logic of Computer Programming," IEEE Transactions on Software Engineering, SE-4, 3, 1978, 199-229.

McKeown, Kathleen R., "Paraphrasing Using Given and New Information in a Question-Answer System," Tech. Report, Dept. of Computer & Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1979.

Miller, Lance A., "Natural Language Procedures: Guides for Programming Language Design," Paper presented at the Annual Meeting of the Association for Computational Linguistics, Georgetown University, Washington, DC, 1977.

Miller, Perry L., "An Adaptive Natural Language System that Listens, Asks, and Learns", in Advance Papers of the Fourth International Joint Conference on Artificial Intelligence, Tbilisi, Georgia, USSR, September 3-8, 1975, 406-413.

Neumann, Peter G., Robert S. Boyer, Richard T. Feiertag, Karl N. Levitt, and Lawrence Robinson, "A Provably Secure Operating System: The System, Its Applications, and Proofs," SRI Project 4332, Final Report, Stanford Research Institute, Menlo Park, CA, 1977.

Parnas, D. L., "On the Design and Development of Program Families," IEEE Transactions on Software Engineering, SE-2, No. 1, March, 1976, pp. 1-8.

Parnas, David L., "Use of Abstract Interfaces in the Development of Software for Embedded Computer Systems," NRL Report 8047, Naval Research Laboratory, Washington, DC, June, 1977.

Ross, D. T., "Structured Analysis (SA): A Language for Communicating Ideas," IEEE Transactions on Software Engineering, SE-3(1), 1977, 16-23.

Roubine, Olivier and Lawrence Robinson, "SPECIAL Reference Manual", Technical Report CSG-45, Stanford Research Institute, Menlo Park, CA, August, 1976.

Sidner, Candace Lee, "Towards a Computational Theory of Definite Anaphora Comprehension in English Discourse," AI-TR 537, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, 1979.

Small, Steven L., "Word Expert Parsing", Proceedings of the 17th Annual Meeting of the Association for Computational Linguistics, San Diego, August, 1979, 9-14.

Teichroew, D. and E. A. Hershey, III, "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems," IEEE Transactions on Software Engineering, SE-3(1), 1977, 41-48.

Thomas, John C., "Quantifiers and Question-Asking," RC 5866 (25388) IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1976.

VanLehn, Kurt A., "Determining the Scope of English Quantifiers," MIT AI-TR-483, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, 1978.

Webber, Bonnie Lynn, "A Formal Approach to Discourse Anaphora," BBN Report 3761, Cambridge, MA, Bolt Beranek and Newman Inc., 1978.

Weischedel, Ralph M., "A New Semantic Computation While Parsing: Presupposition and Entailment," Syntax and Semantics, Volume 11: Presupposition, Choon-Kyu Oh and David A. Dineen (eds.), Academic Press, New York, 1979, 155-182.

Weischedel, Ralph M., "Practical Suggestions for Writing Understandable, Correct Formal Specifications", Technical Report, Dept. of Computer & Information Sciences, University of Delaware, Newark, DE, 1980.

Weischedel, Ralph M. and John Black, "Responding to Potentially Unparsable Sentences," American Journal of Computational Linguistics, 6, 2, 1980.

Weischedel, Ralph M., Wilfried Voge, and Mark James, "An Artificial Intelligence Approach to Language Instruction," Artificial Intelligence, 10, 1978, 225-240.

Woods, W. A., "An Experimental Parsing System for Transition Network Grammars", Natural Language Processing, Randall Rustin (ed.), New York: Algorithmics Press, Inc., 1973, 111-154.

Woods, W. A., "Semantics and Quantification in Natural Language Question Answering", Advances in Computers, Vol. 17, New York: Academic Press, Inc., 1978.

Appendix A

Professional Personnel Associated with the Research Effort

1. Ralph M. Weischedel
   (Principal Investigator)
   Dept. of Computer & Information Sciences

2. Linda Salsburg
   (full-time graduate student fully supported by the contract)
   received the Master of Science in Computer Science (non-thesis option), June, 1980.

3. Arthur W. Mansky
   (graduate student whose computer time for the Master's thesis was supported in part by this contract)
   received the Master of Science in Computer Science, Thesis title: "A Case Study in Natural Language Processing: The RUS System", June, 1980.

Appendix B

Technical Reports to Disseminate Results

At the close of this one-year contract, two technical reports in the Department of Computer & Information Sciences Technical Report Series have been prepared to disseminate results, receive feedback, etc. prior to submission for publication. The references are

Weischedel, Ralph M., "Practical Suggestions for Writing Understandable, Correct Formal Specifications", Technical Report, Dept. of Computer & Information Sciences, University of Delaware, Newark, DE, 1980.

Salsburg, Linda, "A First Iteration at a Mini-set of Features for a Knowledge Representation Language", Technical Report, Dept. of Computer & Information Sciences, University of Delaware, Newark, DE, 1980.