Title: DESIGN, PRINCIPLES FOR ROBUST NATURAL LANGUAGE INTERFACES

CTR project # R6358-0A0 CTR cost sharing # F6358-0A0

Are there existing subprojects? (Y/N) N Is this a subproject? (Y/N) N Main project #

Continuation of project # Type of research RES

Coproject director name

SSN - - Unit

Coproject director name

SSN - - Unit

PROJECT ADMINISTRATION DATA

Administrative data OCA contact INA R. LASHLEY PAD CO IRL 894-4820
Sponsor technical contact PROF. WARREN D PEELE, TREASURER (305) 892 - 6146
Sponsor issuing office MS. JANETTE D. FOSTER (305) 892 - 6146
MGT OFFICE, CENTRAL FLA FACILITY MGT OFFICE, CENTRAL FLA FACILITY
11TH & MASSACHUSETTS AVE., 11TH & MASSACHUSETTS AVE.,
ST CLOUD, FL 32769 ST CLOUD, FL 32769
Security class (U,C,S,TS) U Defense priority rating NA
NA supplemental sheet Equipment title vests with Sponsor GIT

Admin comments -
INITIATION OF PROJECT. *PRIME IS UNDER A DD254. THIS SUBCONTRACT UNCLASSIFIED
SPONSORED PROJECT TERMINATION/CLOSEOUT SHEET

Date 2/24/88

Project No. G-36-673

Includes Subproject No.(s) n/a

Project Director(s) R. E. Cullingford

Sponsor SE CNTR for ELEC ENGR EDUC

Title Design Principles for Robust Natural Language Interfaces

Effective Completion Date: 12/30/87 (Performance) 12/30/87 (Reports)

Grant/Contract Closeout Actions Remaining:

[ ] None

[ ] Final Invoice or Copy of Last Invoice Serving as Final

[ ] Release and Assignment

[ ] Final Report of Inventions and/or Subcontract:
  Patent and Subcontract Questionnaire sent to Project Director

[ ] Govt. Property Inventory & Related Certificate

[ ] Classified Material Certificate

[ ] Other

Continues Project No. ____________________________ Continued by Project No. ________

COPIES TO:

[ ] Project Director
[ ] Research Administrative Network
[ ] Research Property Management
[ ] Accounting
[ ] Procurement/GTRI Supply Services
[ ] Research_Security_Services
[ ] Program Administration Division
[ ] Contract Support Division
[ ] Facilities Management - ERB
[ ] Library
[ ] GTRC
[ ] Project File
[ ] Other
December 31, 1987

Mr. Lou Hoebel  
RADC/COES  
Griffiss AFB, NY  13441  

Dear Lou:

Enclosed is a technical report, "The Design of a Robust Natural Language Interface to a Decision Aid," by Mark Graves and myself. This comprises the final report for our project with you (through SCEEE). Details of the customization of DESI for the BMDA are given in Appendix A. The actual DESI software will be along shortly.

I'm amenable to further consulting with you folks, if some reasonable arrangement can be worked out. Continue to email me at rec@gatech.gatech.edu; I'll be logining in there regularly.

Sincerely,

R.E. Cullingford

Enclosure

c: Contract G36-673 Deliverable Schedule
The Design of a Robust Natural Language Interface to a Decision Aid

R. E. Cullingford and M. A. Graves

School of Information & Computer Science
Georgia Institute of Technology
Atlanta, GA 30319

ABSTRACT

One of the main goals in the design of natural-language interfaces to information-processing systems is robust interaction, the capability on the part of the interface to cope with ill-formed, though quite understandable, user input. In this paper we discuss three design principles to support robust interaction: (a) the need for a "conceptual" level of representation of the user's inputs and the operations and data structures supported by the target system; (b) a "bottom-up" mode of language analysis to permit at least a fragmented representation of problematic user input; and (c) methods of model-directed diagnosis, to allow the repair of those representations and the formation of reasonable target-system queries.

We illustrate these principles by examining the design and implementation of DESI, an experimental natural language interface to a commercial Decision Support System (DSS) for multidimensional numeric databases. We examine the problems which arise when the end-user, a governmental or corporate manager/analyst, is allowed to use free English in place of, or to supplement, the set of commands provided by the DSS. These problems include ungrammatical, misspelled and fragmented input, word-sense disambiguation, ellipsis expansion, and referent selection. The diagnosis process involves recognizing the DSS command(s) intended by the user by using semantic information to type-check and fill in missing command parameters, asking appropriate questions where necessary. The main types of information used are the models maintained of both the DSS database and the prior conversation with the user as these evolve. Experience with real users has shown that our model of diagnosis allows DESI to cope in a reasonable manner with ill-formed inputs, thus greatly enhancing the capabilities of the DSS by incorporating the power of natural language expression.

1.0 Introduction: Coping with Realistic User Input

A primary design goal for natural-language interfaces to database management and expert reasoning systems is robust interaction, the ability to cope with problematic user inputs. What seems to be required for robustness [cf., e.g., HAYE83] is recognizing the class of problem the input typifies; repairing those problems, perhaps by asking questions of an internal model of the user's application or the user himself; and either issuing a reasonable subset of the intended commands to the target system, or failing gracefully with an indication of why the input couldn't be handled.

Let's consider some of these problems in the context of a decision-support tool that provides the usual reporting, question-answering, data-analysis and data-fusion capabilities for corporate databases containing sales records, financial information, etc. We will call such a system the "target system,"
when connected to a language interface. (See Figure 1.) Some commercial examples of decision-support systems are EXPRESS [MDS82], ADDATA [ADS80], ANALECT [DIAL84], STRATEGEM [IP83], and System W [COM83].

The problems with real user input include: (a) agrammaticality; (b) word-meaning selection; and (c) discourse phenomena such as ellipsis and anaphora. The term "agrammaticality" is intended to cover several classes of problems. First, of course, is outright ungrammaticality, the failure of an input to conform to a pre-specified grammar describing the acceptable forms of input. In (1), for example, we see a failure of the user to adhere to the convention called do support for certain kinds of wh-questions:

(1)  
what data i have

Note that, strictly speaking, (1) also has morphological (capitalization) and punctuation problems.

Agrammatical inputs may also be telegraphic, that is, they may omit any non-essential determiners, modifiers, conjunctions, etc., in an effort to get to an extremely concise form:

(2)  
got bos and nyc, merge on measure, print

In this input, the user is asking for a retrieval of some data ("the data for the Boston and NYC markets") from some implicit data source, a merger of that data along some "dimension" of the data model, and a "display" of the combined data [Note 1]. The user has discarded all but the essential action and object words.

Agrammatical inputs may also contain misspellings. The following contains run-on words, and a word with transposed letters:

(3)  
dosame with wcoast data, then svac

In (3), the user is asking for a repeat of a prior command sequence on some different data, followed by

1. Exactly what the "retrieve," "merge," and "display" operators do is not important for the preliminary discussion. We will discuss a concrete decision-support system in Section 2.0
"permanent" storage of the result. Of course, some words that look misspelled, in the sense that they aren't in the interface's lexicon, may actually be names that the user wishes to attach to some data objects:

(4) place result in bnoz

The interface needs to be able to judge when an unknown word is intended as a name.

A particularly insidious species of misspelling occurs when a word is botched in such a way as to appear to be a legal, known word [e.g., Wies85]:

(5) load store

Here the user may have intended the retrieval of a data object called "stores," "ustore," etc., or a retrieval followed by an immediate save of some unnamed object. The latter, of course, is an unlikely reading on pragmatic grounds.

A related phenomenon occurs when the user types in a pre-existing word (e.g., a command or object name) in a position which seems to be indicating a name:

(6) place sum of fool and foo2 into save

Suppose "save" is a system keyword corresponding to an act of permanent data storage. Should the interface, recognizing the attempt to use "save" as a name, prevent the user from working with a keyword this way? Or should it silently make the word "save" ambiguous?

This latter possibility brings up the problem of word-meaning selection in the form of word-sense ambiguity. It is often claimed that the meaning-ambiguity problem can be finessed if the application is carefully constrained. Examination of how real users work with interfaces, however, indicates that the problem, while reduced in scope, still remains. It is especially acute with respect to difficult words, such as "give," "take," "get," "put," etc., which have so many meanings in the general language as to almost meaningless by themselves. Consider, for example:

(7)
put result into db
put average of bnoz for product into bnozl

(8)
get northeast and middatl
get bos, hartford and nyc from northeast

(9)
take 0.5 of middatl plus northeast
take bos and nyc out of northeast and extend with middatl

(10)
show dimensions of northeast
' show middatl averaged over time for first quarter, then halved

In the sentence-pair (7), "put" first indicates an permanent storage event, then a data movement operation placing the output of an arithmetic operation into a named object. The pair (8) shows "get" first being used to retrieve the named objects "northeast" and "middatl" from permanent storage; then used to specify the extraction of a subset of the data from the object northeast. Example (9) shows "take" in the senses of "apply a certain arithmetic operation" to an object, and "extract some data from an object." Finally, Example (10) illustrates the difficult word "show," itself, used in the senses of "display the internal structure of a data object" and "display the contents of a data object formed by certain operations." (These senses usually correspond to different target system commands in decision-support systems having these kinds of capabilities.) Note that the operations intended in the second sentence of (10) are ambiguous. Is it "Show middatl averaged...then halved," or "Show middatl averaged...then show middatl halved"? A robust interface needs to preserve the alternative readings of a truly ambiguous input, and ask the user to decide later.

A third class of language-analysis problems is caused by discourse phenomena, such as ellipsis and anaphoric reference. The term "discourse" is used because the problem solution usually involves use of information outside the current clause, that is, reference to the prior course of the conversation between the user and the target system. In ellipsis, for example, the user provides only a fragment of a complete proposition; reference to a prior input is expected to provide the complete structure in which the fragment is to be imbedded. For example, an input such as:

(11a)
print current for wcoast, first and second quarters of 85,
and w1 w2 w3

followed by:

(11b)
now for FY84

indicates the user's desire for a repeat of the "display" command, with the corresponding specifications
in the original ("first and second quarters of 85") replaced by the ellipsed data ("for FY84"). Note that
this correspondence must be inferred on semantic grounds, i.e., using the underlying data model.

Ellipsis need not be restricted to objects that must be imbedded in actions or queries, but may
require the reconstruction of prior sequences of actions. For example, the sequence:

(12)
extend persdata with hires, average over salary, and display.
repeat for 83data and 83hires.

requires mapping the ellipsed information ("83data and 83hires") into the appropriate place(s) in the
previous command sequence, here into the "extend" command. Then the sequence must be repeated in
the correct way.

Another discourse effect which must be handled is anaphoric reference resolution, the selection
of a referent(s) for a pronoun or definite description from those mentioned previously. For example,
given:

(13a)
load hires and persdata

followed by:

(13b)
show them

or even:

(13c)
show

the interface needs to be able to find the correct antecedent of "them" -- implicit in (13c). Often the
process of referent resolution will interact with the word-sense disambiguation task described above.

For example, in:
what objects are in workspace? show them.

the identification of "them" with "objects in the workspace" should cause the disambiguator to select the "display object data" sense of "show." On the other hand,

how many dimensions does persdata have? show them.

should cause the selection of "display object structure."

1.1 Design Principles for Robust Interaction

The problems illustrated above are difficult, indeed, and no existing language interface can claim to handle all of them in all their forms. Our experience with interfaces intended for real (including casual, infrequent and non-technical) users of real decision-support systems operating on real databases does, however, indicate that certain design principles can, if adhered to, afford a good degree of robustness of interaction. We sketch out these principles in succeeding sections, before turning to their application in an experimental implementation called DESI (DEcision Support Interface), a front end to a commercial DSS called ANALECT [DIAL84].

1.1.1 "Conceptual"-Level Representation

It's clear that robust interaction requires that the interface "understand" the user's input in a reasonably deep sense. That is, the system should deal internally with a representation of the meaning of the user's input sentence, rather than with the sentence itself or with a description of its syntactic structure. The main advantages of this are: 1) the system deals with a "canonical" meaning structure (supplied to it by the analyzer), rather than with the potentially many sentences having the same meaning; 2) the inference and search processes that find the command being referred to and check on the validity of supplied command parameters can be keyed directly (in a procedurally imbedded manner) to parts of the meaning structure; and 3) inferencing can interrogate an internal model of the user's database and the effects of the DSS operations.
There are a number of representational schemes which allow reasonable conceptual-level structures to be developed for sentence meanings, for example, KL-ONE [BRAC78], KRL [BOBR77], Active Structural Networks [NORM75], Preference Semantics [WILK75], Commonsense Algorithms [RIEG76], Conceptual Dependency [SCHA75], Semantic Network representations [e.g., WOOD75; HEND76; BRAC78; and STEF80], and various proposals based on the First-Order Predicate Calculus [e.g., MCCA77; HAYE77; and FILM79].

We have taken the Conceptual Dependency/Preference Semantics approach that assigns an underlying primitive predicate with characteristic case-frame as the basis for the representation of words that are very similar in meaning. For example, the underlying concept of a "physical transfer" is common to words such as "go", "come", "take", "walk", "throw", etc. The differences in meaning are accounted for by assigning slightly different representations to various cases in the case frame. Concrete examples of the representations used by our experimental system, DESI, are discussed in Section 3.

1.1.3 Bottom-Up Language Analysis

The language analyzer used by DESI is one of a class of analysis programs called conceptual analyzers. These are programs which attempt to map an input string directly into a meaning representation, using whatever morphological, syntactic, semantic, contextual, etc., cues are available. They are distinguished from other types of analysis programs in that they do not attempt to first analyze the input syntactically, then assign a semantic reading to the syntactic structure [e.g., WOOD70, MARC80]. Nor do they conduct a simultaneous syntactic and semantic analysis [e.g., WINO72, BROW75]. (Such analyzers are usually called "parsers.") Syntactic features such as word order and noun-group constituency are used by a conceptual analyzer only to guide the meaning-extraction process.

The basic mechanism used by DESI’s analyzer is a predictive one which attempts to provide expectations about what will be read on the basis of what has already been read. The distinctive feature of the approach is that analysis is not driven from the top down, as in many explicit-grammar approaches, but from the bottom up, by the lexicon.
Word definitions describing the meaning structure(s) built by a word are kept "off-line" in a dictionary, and are not called into active memory until the word is actually seen in the input stream. Expectations associated with a word definition are encoded in a special type of production rule (test-action pair [NEWE72]) called a request [RIES75]. Requests are activated when the associated word definition is loaded, i.e., placed in a short-term memory of requests to be considered. The request consideration process repeatedly selects a request and evaluates its test part. If the test returns "non-nil," the request is said to have fired, its action part is evaluated, and it is removed from the request memory. Requests can check semantic, lexical, or contextual features of the run-time environment, and create or interconnect conceptual structures. Moreover, they can cause other requests to be activated or deleted.

The most important advantage of such an approach for robustness is that the words actually create their associated meaning structures as they are read. Thus, in ungrammatical cases, fragments of meaning tend to be left over when the analysis process terminates. We will discuss these ideas in more detail in Section 4.

1.1.3 Model-Directed Diagnosis

For various kinds of ill-formed inputs, it is often possible to discern what the user wants by heuristically recognizing and repairing the problems. DESI's diagnostician takes a problematic input and applies various heuristic rules, for example, to fit together fragments of meaning produced by the analyzer. The diagnosis process may need to ask questions of an internal model of the user's application, of the ongoing conversation, or of the user himself.

Suppose, for example, the user types "what data i have." At the end of analysis, there would be two fragments, one representing the concept "what data," the other representing the ambiguous readings of "i have." This particular pattern of fragments suggests that a syntactic convention (in this case, do-support) has been violated. The rule that would be applied is the "ignore-syntax-rule," which puts the analyzer into a semantics-only processing mode. As it happens, one of the senses of "have" accepts the "data" concept, thus allowing a complete, recognizable meaning structure to be formed. We will give more details of this diagnosis process in Section 4.
1.2 Outline of the Paper

To make the subsequent discussion concrete, we will describe a natural-language interface, DESI, to a commercial decision-support system, ANALECT [DIAL84]. DESI was designed with the three criteria just described in mind. Thus, although it has not yet been extensively tested in real applications, it can provide some support for our claims concerning robustness.

Section 2 gives an overview of ANALECT, and in particular its multi-dimensional analysis (MDA) component, for which DESI creates queries. This section also gives some sample interactions with DESI. Section 3 is a discussion of knowledge representation issues for DSSs of this sort, and of the style of language analysis used in DESI. The nature of this mode of analysis is such that it terminates with fragments of meaning for ill-formed sentences. Section 4 discusses a process of model-directed diagnosis, which attempts to categorize the problem exemplified by a given set of fragments. Finally, Section 5 comments on the current status of this system, and on its implications for the design of future interfaces.

2.0 System Overview and Sample Conversations

This section is an overview of the natural language interface (DESI) to the multidimensional analysis (MDA) component of a comprehensive DSS (ANALECT). First, we introduce some terminology which is useful for describing DSS's of the sort we are interested in. Then we give a brief overview of the major components of DESI. Finally we give some examples of a conversation between a user and the DSS, mediated by the interface.

The words and phrases that appear in user inputs to DESI are normally descriptive of the user's objects, multidimensional associations of numeric data (marketing or financial data, for example) arranged in a time-series, and the operations the user wishes to carry out on them using ANALECT. The objects which are to be manipulated in an ANALECT operation are called source objects (of which there may be one or two), and the result of the operation is placed in a target object, which may be the same as one of the sources. The objects being manipulated will often be named. However, the system
maintains an idea of the "current" object, which can be one of the implicit source objects to the next command.

As stored in an ANALECT object, this data has a number of primary dimensions, the major descriptors of the object's organization. For example, a marketing-sales object might contain dimensions for "product," the collection of things for which sales data exists; "units," whether the data is given in terms of monetary units or simply number of products sold; "market," places where the products were sold; and "time," describing the intervals during which the products were sold. The dimensions group together dimension values, the constant terms which, when one is selected from each dimension, uniquely define an item, one of the numeric values in the database object. The dimension MARKET for example, could consist of dimension values standing for the cities in which the products are sold: ATLANTA, HOUSTON, SEATTLE, etc. The dimension values can also be organized hierarchically. For example, it can be declared to DESI that the term NORTHEAST is to stand for the collection of dimension values on the MARKET dimension consisting of NY, BOSTON and PHILADELPHIA.

The objects created by the user as the DSS runs can be stored in two places: the database, which is the long-term repository for objects the user wishes to keep across sessions; and the workspace, into which some of the database objects are loaded when the user wishes to work with ANALECT. New objects created in the workspace by ANALECT operations can be placed in the database for permanent storage.

As the block diagram of Figure 2 indicates, DESI has four major subsystems, all of which are based on existing products of artificial intelligence research:

1. A natural-language analyzer (adapted from the one described in [CULL84]), which processes typed English input into a representation of its meaning. The representation scheme used is a simple version of conceptual dependency [SCHA75], based on the database entities discussed above.

2. A recognition and diagnostic package [BOOT85], which is intended to examine the analyzer's output, identify the DSS command(s) being referred to, and correct or fill in any parameters needed to make a well-formed command. Ellipsis expansion, referent selection, fragmentary output and unknown words are handled here. (At this point, both the command and its parameters are still in a "conceptual" form).

3. A restricted-form natural language generator (very close to the one described in [CULL82]), which is given the output of the recognition subsystem and generates a DSS query.
4. A knowledge-base management and retrieval subsystem [CULL83], which is used to quickly find associations between conceptual forms and data/program packages (the "words" of the DSS query language in the case of the generator, and the diagnosis functions associated with each of the DSS commands in the case of the recognition subsystem).

The first two of these modules will be discussed further below, since they are responsible for robust interaction with the user. To give the reader some feeling for the types of conversations a manager/analyst can have with DESI and the DSS, we present here a sample of input/output for two short sessions with the system, edited slightly for readability. Lowercase sentences are user input. Uppercase output is the query to the DSS. (The actual DSS output is not shown; it is typically a paginated printout of the contents of the target object of the DSS command, organized by its dimensional structure.)

The first session shows the user working with some previously stored objects. After logging in to DESI, the user first needs to find out what the database contains. (Database objects and vocabulary defined in previous sessions are maintained by DESI on a per-user basis.)

User:
what objects are there

DESI:
The workspace is empty

The database contains an ACCESS-type object with dimensions:
MEASURE, MARKET, PRODUCT and TIME

You can RETRIEVE the complete object by typing 'RETRIEVE ACCESS'

Stored objects:
EX3 EX2 EX1 COMB INV1

Here the user's question has been handled directly by DESI, which interrogates its internal representation of the user's data objects. The user workspace currently contains nothing (the session has just started); the global database, conventionally named ACCESS, has the the dimensions shown; objects created by the user in the workspace in prior sessions and then placed back into the database are shown next.
User:
get w1 w2 and w3 in NY and Chi for 7901, 7902, and 7903,
    for both dollars and units, and call that data smalldata

DESI Output:
RET FFOBJ0 PRODUCT W1 W2 W3 MARKET NY CHI TIME 7901 7902 7903
    MEASURE D U

DESI Output:
RENAME FFOBJ0 TO SMALLDATA

Here the user has asked for a subset of the data in the ACCESS object (specified by the dimension
values given) to be loaded into the workspace in the form of an object named SMALLDATA. DESI
handles this by a combination of a RETrieve command, whose target is an object with made-up name
FFOBJ0, followed by a RENAME of that object to the desired name. (DESI could have eliminated the
RENAME by making SMALLDATA the target of the RET. Such optimizations are not in the system as
of this writing.)

-----------------------------------------------------

get w4 and w5 as indicated above and name it mediumdata

RET FFOBJ1 PRODUCT W4 W5 MARKET NY CHI TIME 7901 7902 7903
    MEASURE D U

RENAME FFOBJ1 TO MEDIUMDATA

Here the "as indicated above" is analyzed as shorthand for "redo the last sequence of commands, with
the new specifications overwriting the corresponding ones in the old sequence." The referent for "it" is
resolved as the target of the RET command.

-----------------------------------------------------

combine smalldata and mediumdata into groupdata

MERGE SMALLDATA WITH MEDIUMDATA FORMING GROUPDATA

"Combine" is a way of referring to the DSS MERGE command. (It can also mean EXTEND, as in
"combine FOO and BAZ along dimension BAR."
DESI:
The dimensions of object GROUPDATA are:
PRODUCT MARKET TIME MEASURE
More? (y/n)

Dimension PRODUCT has values:
W1 W2 W3 W4 W5

Dimension MARKET has values:
NY CHI

Dimension TIME has values:
7901 7902 7903

Dimension MEASURE has values:
D U

Do you want to see the object’s values? (y/n)
User:
y

DESI Output:
PRT GROUPDATA

DESI interprets "display" as a request for a detailed listing of the attributes and values of the last target object. The attribute display is handled by DESI directly. The values are shown by use of the DSS PRT command.

sum smalldata over product and call it totsmall

COLLAPSE FFOBJ3 = SMALLDATA SUM OVER PRODUCT

RENAME FFOBJ3 TO TOTSMALL

This is an example of an arithmetic operation being applied along one dimension of an object, yielding the reduced object TOTSMALL.

get from smalldata w1 dollars from 7903 and put it in w17903
SELECT FFOBJ4 FROM SMALLDATA WHERE MEASURE EQ D AND PRODUCT EQ W1 AND TIME EQ 7903

COMPUTE W17903 = FFOBJ4

Note that "get" is here interpreted by DESI as meaning a selection of some data from a workspace object, rather than a RETRIEVE, as above. The noun group "smalldata w1 dollars" has been resolved into an object and specifications along separate dimensions. The COMPUTE could have been replaced by a RENAME, or eliminated altogether.

get same from totsmall and call it sm7903

SELECT FFOBJ5 FROM TOTSMALL WHERE MEASURE EQ D AND PRODUCT EQ W1 AND TIME EQ 7903

RENAME FFOBJ5 TO SM7903

"Same" has been interpreted in a manner analogous to "as indicated above" in a prior input.

divide w17903 by sm7903 and print result

COMPUTE FFOBJ6 = W17903 / SM7903

PRT FFOBJ6

Here we have an example of an arithmetic operation with two objects as arguments. The analyst is apparently interested in the ratio between dollar sales of a given product in a given time period for all markets versus sales for all products of a given type.

show all objects in workspace

Workspace objects:
FFOBJ6 SM7903 W17903 FFOBJ4 TOTSMALL GROUPDATA MEDIUMDATA SMALLDATA

change the name of sm7903 to march79

RENAME SM7903 TO MARCH79
Here the analyst looks at the objects that currently reside in the workspace, and gives one of them a more meaningful name.

average march79 over market and multiply the result by 1.2 and put answer into march80

COLLAPSE FFOBJ8 = MARCH79 AVG OVER MARKET

COMPUTE FFOBJ7 = FFOBJ8 * 1.2

COMPUTE MARCH80 = FFOBJ7

This is a typical example of a "chain" of DSS commands being specified by a (not particularly grammatical) coordinate construction. DESI contains a recognition package that deals with constructions of this sort by recursively calling the recognizer on the individual component commands. The analyst seems to be making a sales growth prediction.

User:
show values for march79 and march80

DESI:
The dimensions of object MARCH79 are:
PRODUCT MARKET TIME MEASURE
More? (y/n)

The dimensions of object MARCH80 are:
PRODUCT MARKET TIME MEASURE
More? (y/n)

DESI Output:
PRT MARCH79

PRT MARCH80

User looks at his growth comparison.

get from mediumdata all values gt 100 and prt
SELECT FFOBJ9 FROM MEDIUMDATA WHERE VALUE GT 100

PRT FFOBJ9

Here we see another use of "get" as a SELECTion operator, with an arithmetic comparison specified for the resultant object. The implied target object of the PRT is correctly supplied.

save the following objects: smalldata, mediumdata, march79 and march80
STORE SMALLDATA MEDIUMDATA MARCH79 MARCH80

Finally, the user stores the "interesting" objects back into the database, for possible use in future sessions.

The user's inputs in the above conversation, while mildly ungrammatical in places, were not particularly problematical. The next conversation illustrates some more severe problems.

User: what data i have
DESI:
The workspace is empty

The database contains an ACCESS-type object with dimensions: MEASURE, MARKET, PRODUCT and TIME

You can RETRIEVE the complete object by typing 'RETRIEVE ACCESS'

Stored objects:
EX3 EX2 EX1 COMB INV1

Here the interface has had to cope with the missing auxiliary "do." The analysis phase terminates with two meaning fragments: one for the questioned objects ("what data"), and one representing the alternative meanings of "have." The diagnostician applies the ignore-syntax heuristic, and restarts the analyzer. Now "have" finds an appropriate "object," and selects the "user-possess-data" sense of "have."

User: let wcoast refer to pord-losd
DESI: EQUATE WCOAST TO PORT-LOS
User: get the nielsen share of market
for 6400 in mqf84 in wcoast and print
DESI:
RET CURRENT MEASURE N.SOM MARKET
  WCOAST PRODUCT 6400 TIME MQF84
PRT CURRENT

User: name it wcmq84
DESI: RENAME CURRENT TO WCMQ84

The first user command sets up a synonym between the lexical item "WCOAST" and a set of markets
on the west coast. (As the session terminates, DESI will ask whether this synonym should be be made
a permanent part of this user’s lexicon.) Then, the user specifies that set of markets as part of a
RETrieve of a subset of the data in the global database. DESI maintains a large phrasal database
which allows the user to refer descriptively to an objects dimensions, so it’s not necessary to remember
cryptic words such as "N.SOM." The object CURRENT, which is the current focus of attention, is
specified as the target of both the RETrieve and PR(in)T commands. (DESI handles complicated coor-
dinate constructions as sequences of the underlying MDA commands.) DESI also resolves referents for
pronouns such as "it" by looking at the its record of the conversational exchanges. This allows the
disambiguation of words such as "name."

User: print the weighted value share
of 0900 in pot for january through
march of 84.
DESI:
By POT, do you mean:
  1) PORT
  2) PUT
  3) none of the above
(Please type the number of the correct
item)
User:
1

DESI:
RET CURRENT MEASURE WVS MARKET PORT
  PRODUCT 0900 TIME JAN84-MAR84
PRT CURRENT
DESI:
Do you want POT to be a synonym for
  PORT? (Answer Y or N)
User:
DESI performs a spelling-correction operation on the word “pot,” and asks the user to indicate which alternative is meant. Later, it asks whether the misspelled word is to be remembered as a synonym. DESI “notices” that an object with the required structure is not in the workspace, so it loads the necessary data. Note how the specification “January through March of 84” has been mapped into the correct sequence of values on the TIME dimension.

User: add to wcmq84, sum over geography, divide by 10, show result

DESI:
MERGE CURRENT WITH WCMQ84 FORMING CURRENT
COLLAPSE CURRENT = CURRENT SUM OVER MARKET
COMPUTE CURRENT = CURRENT / 10.
PRT CURRENT

Here the user telegraphically instructs the DSS to form the merger of the data in two objects. (DESI correctly disambiguates “add,” and selects CURRENT as the second source object.) Then the user performs a calculation along one dimension of the resulting object. Such computations are typical of planning or war-gaming sessions with a DSS.

User: avg

DESI:
Sorry, couldn’t decide on the meaning of “avg”
  1. The command AVG
  2. The dimension AVG
Please retype your command.

Here the user has given a truly ambiguous input: AVG can either be an averaging computation or a dimension in an object that has been averaged. DESI attempts to explain its confusion to the user.

User: delete everything but current

DESI:
Here, DESI has handled the negated quantifier phrase "everything but" and gotten rid of the appropriate object. The analyzer/recognizer has a limited ability to deal with quantified inputs.

User: save as wc84
DESI: RENAME CURRENT TO WC84 STORE WC84

Finally, the user indicates that the result of the session is to be saved. DESI selects the correct object, after changing its name. Note that DESI has had to "reach back" to the last correctly computed object, CURRENT.

3.0 Meaning Representation and the Wordsense Database

As mentioned in Section 1, DESI works with meaning representations for sentences that are based on a primitive predicate with characteristic case-frame. The idea is to assign similar representations to words that are very similar in meaning. For example, the underlying concept of a "mental transfer of information" is common to words such as "talk", "chatter", "announce", "admit", "deny", etc. The differences in meaning are accounted for by assigning slightly different representations to various cases in the case frame.

The meaning structures manipulated by the system are recursive, slot-filter structures [see, e. g., CHAR80], based on a predicate with labelled arguments. For example, the sentence "object FOO is in the database" is represented as:

(*)
(p-config confrel (inside)
 con1 (series seriestag (FOO))
 con2 (container conttype (*DB))
 mode (modes model (:t))
 time (times time1 (:pres)))

Here, the primitive predicate is p-config (physical configurational), which expresses a stative of physical relation (location, contact, etc.) between two entities. One of the entities, a time-series object named
FOO, is imbedded in the structure at the argument labelled conl. The other is the unique system "container" of information, the *DB. Note that the object and container have recursively the same structure as the p-config. (The mode and time labels in (*) indicate that the concept is asserted and has the present tense, respectively.)

Sentences of slightly different meaning get slightly different structures. For example, "what's the database contain?" would have a structure identical to (*), except that the conl label would be filled with

(entity ref (*Q*))

that is, by an entity whose referent is sought.

The representation scheme is simplified for this application by the fact that there are only six different kinds of entities: containers, objects, dimensions, dimension values ("dimvals," for short), numerical database items, and time-durations, which define the basic dimension in a time-series object. Furthermore, the only actions that the system needs to know about are the ANALECT MDA commands themselves, of which there are about twenty. The most important stative primitives are associated with the immediate results of these actions. For example, the result of a "RET FOO" command would be expressed by a p-config such as (*), with the conl label filled with

(container conttype (*WS))

that is, the workspace.

The conceptual analyzer and generator which respectively give forms like these to the recognizer, and express its output to the DSS, share a database of wordsenses, associations between symbols (lexemes, words) and their underlying "conceptual" representation. For example, one of the senses of the word "combine" refers to the ANALECT command EXTEND, by which one object gets additional data from another along a dimension shared by both. The wordsense, called wsCOMBINE1, is declared as follows:

(def-wordsense wsCOMBINE1
  surface-form (combine)
  ws-structure)
(FFcommand-f commtype (EXTEND)
  source1 (nil)
  source2 (nil)
  target (nil)
  dim1 (nil))
constraints
  ((source1 couldbe-series-p)
   (source2 couldbe-series-p)
   (target couldbe-series-p)
   (dim1 couldbe-dimension-p)))

The record macro def-wordsense manages the wordsense database. The "surface-form" associated with the wordsense is the word or phrase that expresses the concept. The "ws-structure" is an evaluable form which returns a new instance of the wordsense. (Note that the same word may be used more than once in a session.) Here, the form is based upon the primitive type FFcommand, which is the canonical ANALECT command. ("FFcommand" stands for "FireFly command" for the FireFly module, which is the multidimensional analysis component of ANALECT.) The particular command is an EXTEND, with labels source1, source2, target and dim1, initially with empty (nil) fillers.

The "constraints" are predicate expressions which can be used to check validity of proposed arguments. For example, the label dim1 requires an argument which "could be" a dimension. (This can be used to select the referent of a word such as "it.") The wordsense wsCOMBINE2 expresses the MERGE sense of "combine," in which a new object which is a merger of all the data of the source objects is created. All the wordsenses having the same surface form are combined into a dictionary definition for the form which is ambiguous, as described below.

4.0 Conceptual Analysis Subsystem

Most existing natural language front ends base their processing on an explicit grammar. Such front ends have been reasonably successful as general-purpose, transportable interfaces to traditional database management systems [cf., e.g., TOIS85]. The nature of grammatically driven processing leads to certain problems, however, in tasks of a more informal, conversational nature.

First of all, an explicit grammar tends to make the system's computation excessively top-down. Thus, it is usually not particularly robust under deviant (e.g., ungrammatical, telegraphic or ellipsed) input. Work on adapting grammars to handle various classes of variant input has been done [e.g.,
KWAS79, CARB83], but the basic problem, that the system will not accept non-conforming inputs, remains.

A second, more severe, shortcoming of existing front ends is that, because they are intended to be general purpose, they typically have no well developed conceptual model of their application. Commercial interfaces usually work with general-purpose database management systems. Knowledge of the domain, e.g., payroll accounting or inventory control, is only implicit in the choices of the vocabulary that are made. As a result, the interface has no source of reasoning power to assist it with the myriad of extremely difficult problems that realistic user inputs, even well-formed ones, cause.

In this paper, we argue for an alternative style of language analysis in which the focus of processing for both input and output is at the level of the lexicon, i.e., the words and phrases of a language, rather than its grammar. (See [CULL86] for a more thorough discussion of the advantages of this approach.)

Language analysis, in particular, has a very strong bottom-up nature in the approach to be described. Such analyzers [e.g., RIES75, BIRN81, CULL84, DYER84] tend to produce fragmented meaning structures for ungrammatical or ellipsed inputs. Thus, there is the possibility for diagnosis of the fragments, in order to determine a reasonable reading of the input [BOOT85].

The language analyzer used by DESI is one of a class of analysis programs called conceptual analyzers. These are programs which attempt to map an input string directly into a meaning representation, using whatever morphological, syntactic, semantic, contextual, etc., cues are available. The basic mechanism used by DESI's analyzer is a predictive one which attempts to provide expectations about what will be read on the basis of what has already been read. Word definitions describing the meaning structure(s) built by a word are stored in the dictionary format described in the last section, and are called into active memory when the word is actually seen in the input stream.

As mentioned in Section 1, expectations associated with a word definition are encoded in a kind of production rule called a request [RIES75]. Requests are activated when the associated word definition is loaded, i.e., placed in a short-term memory of requests to be considered. The request consideration process repeatedly selects a request and evaluates its test part. If the test returns "non-nil,"
the request is said to have fired, its action part is evaluated, and it is removed from the request memory. Requests can check semantic, lexical, or contextual features of the run-time environment, and create or interconnect conceptual structures. Moreover, they can cause other requests to be activated or deleted.

Associated with a meaning structure built by a word are a set of slots and a set of expectations embodied in requests indicating how roles are to be filled. Consider, for example, the conceptual analysis of the sentence "John ate an apple." (We will ignore the details of how morphology and noun-grouping are managed; see [CULL86] for an extensive discussion of this.) The dictionary definition for "John" might look like this:

```
(def-word John
  pos noun
  toprcq
  ((test t)
   (actions
    (add-word-con wsJOHN1))))
```

```
(def-wordsense wsJOHN1
  surface-form (John)
  ws-structure (person-f persname (John)))
```

Here, def-word is a record macro for managing the analyzer's dictionary. "John" has Part Of Speech "noun." The "top" request for "John" has a test part which always evaluates to non-nil, so the action part is always executed. The single action given here calls the function add-word-con to add the meaning structure for the associated wordsense (which defines a named person as shown) to the short-term memory of available concepts called the C-LIST. The word "apple" would contain a similar definition, except that the structure added to the C-LIST would be based upon the primitive ingobj, or ingestible object.

Neither of these words activates any further requests. The word "eat," however, does:

```
(def-word eat
  pos verb
  toprcq
  ((test (= topcon (add-word-con wsEAT1)))
   (actions
    (activate
     (actspot-req topcon (actor) couldbe-person-p)
     (objspot-req topcon (obj) couldbe-ingobj-p))))
```

Here, "eat" first adds the structure defined by the wordsense to the C-LIST, then calls the function *activate* to add two new requests to request memory. (The function := is the *run-time binding operator*, which propagates the actual name of the concept returned by *add-word-con* for this instance of "eat" into the body of the request. All instances of "topcon" will be replaced by that name. Since := always returns non-nil when used in this way, we get the effect of (test t) as in the definitions of "John" and "apple.")

The first request, named *actspot-req*, embodies a test which seeks a person (using *couldbe-person-p*) to fill the *actor* slot in the concept built by "eat." The place where the request looks for its actor is in the "act spot," that is, among the C-LIST items preceding the concept for "eat" in the active voice, in a prepositional phrase governed by "by" in the passive. The second request, named *obispot-req*, seeks an ingestible to fill the *obj* slot in the *ingest* structure, following that structure if the voice is active, preceding it if passive. (Both of these requests implement a fairly complicated symbolic computing activity. See [CULL84] for more details.) This simple (looking) request cluster thus embodies the syntax and semantics of "eat."

The analyzer, proceeding right-to-left through the sentence, first activates and fires the request for "John," adding the associated structure to the C-LIST. Next, the top request for "eat" adds the *ingest*, and queues up the two requests. The *actspot-req* fires immediately, since "John" is already available. The concept is inserted into the *ingest* as the *actor*, and the "John" concept disappears from the C-LIST. The *obispot-req* is a prediction, in this case, and does not fire when first considered. Next the topreq for "apple" fires (ignoring the determiner "an"), and the *ingobj* gets placed on the C-LIST. Now the second request for "eat" fires, and the "apple" concept becomes the *obj* filler in the *ingest*. The process terminates, leaving a well-formed conceptual structure as its result.

More complicated things happen when the words have several meanings, as for example is the case with "combine." If we restrict attention to the imperative form only, its definition is
Here, a structure (called a vel) is added to the C-LIST which represents the two-ways ambiguous concept. The two calls to := propagate the names for the subcons representing the individual meanings through the rest of the request through the names "extendcon" and "mergecon." (The function grf, or get-role-filler, extracts the filler associated with the slot (role) named by its first argument from the structure named by its second.)

The first call to activate creates requests to examine the C-LIST for features which would allow a decision about disambiguation to be made. The first, a velprepspot-req, searches for a dimension concept associated with a prepositional constituent governed by "along," "for" or "in." If it finds one, it compresses the ambiguous concept down to the EXTEND concept, and inserts the dimension concept at slot dim1. Usually, all the requests that were activated at the same time are suppressed at this point. The second request, using the predicate clp-p, looks for a following clausepoint, an indicator (e. g., the end of the sentence) that the clause associated with "combine" has ended. If a clausepoint is reached without a dimension’s being found, the concept is compressed to a MERGE. Since there are arrangements for staged compression steps (i. e., a 3-ways ambiguous concept may be compressed to a 2-ways ambiguous one), the computation can get quite complex.

This, in necessarily broad outline, is how DESI’s basically bottom-up analysis process proceeds. The system is built on top of several dozen named requests such as actspot-req and velprepspot-req, together with about twenty specialized actions other than activate. Dictionary definitions are easily and
flexibly created once the basic ground rules are understood. Due to the extreme simplicity of the meaning representation scheme, very few unexpected interactions have occurred in DESI between existing vocabulary and new words as they are added.

5.0 The Diagnostician

In DESI, understanding the user's input involves recognizing which MDA operations have been requested, and generating well-formed queries. The module which carries out these tasks, the model-directed diagnostician, is responsible for robustly coping with a user's inputs.

To accomplish this, the diagnostician must have information about each command: what its function is, what information is needed to execute it, and what its command syntax is. In addition, the diagnostician must know the current state of the database environment, how each command is affected by that environment and the effect of each command. Using this information, it can determine that the user's specifications are "legal," i.e., consistent with the current state of the database and suitable to the command in question.

Command parameters not explicitly given by the user can often be inferred from the database models, or if that fails, asked directly of the user. For example, the request "print new york and boston," would generate:

PRT CURRENT WHERE MARKET EQ NY OR MARKET EQ BOS

Note that the name of the dimension NY and BOS are members of, i.e., MARKET, has been filled in by DESI. Note, also, that the literal "and" in the input has been mapped into the logical "OR" required by the database.

Robust handling of a user's request is thus directly dependent on successfully "recognizing" the MDA operations that must be executed. This is by no means an easy process. The diagnostician's input is the meaning structure(s) for the user's input, as produced by the conceptual analyzer discussed in the last section. In the simplest case, the analyzer produces a well-formed structure which clearly indicates the commands in question. But if the user uses words unknown to the system, or uses a word in an unanticipated manner, or gives an ungrammatical input, the analyzer's output will be fragmented.
i.e., a list of meaning sub-concepts rather than a single well-formed one.

Note that even well-formed output from the analyzer does not guarantee immediate recognition. For example, the ellipsed input, "for new york and boston," will produce a well-formed meaning representation, but by itself gives no indication of what command is being referred to. In such cases, examination of prior context, e.g., the conceptual form corresponding to prior queries issued to the DSS, is necessary to identify the command being requested.

5.1 The Recognition Phase

All knowledge needed to process a command is command dependent. In DESI, this knowledge is incorporated into command specific procedures which examine and modify the concepts produced by the analyzer into well-formed command concepts. It is the responsibility of the "recognizer" to identify the MDA command(s) associated with the user's request, in order to gain access to this procedural information.

DESI associates a "package" of command-specific information with each command. Each package contains all the information needed to process a particular command. This includes: a set of conceptual patterns designed to match the characteristic ways of referring to the command; a conceptual template to be filled in with command parameters and sent to the generator; an optional match-time function used to place restrictions on the matching process; a diagnostic function encoding the semantic information needed to process the command; and a run-time function responsible for updating the database and conversational models maintained by the system. Extending the system to incorporate new commands is simply a matter of setting up new command packages for each command.

Symbolic pattern matching is the mechanism used for recognizing a user's request. The command patterns are matched against the concepts produced by the analyzer until successful. If necessary, the optional "match" functions are employed. A successful match determines the appropriate MDA command.

The feasibility of this approach is dependent on the number of commands which define the MDA system, and the number of patterns needed per command for recognition purposes. DESI has to recog-
nize approximately twenty MDA commands, a relatively small number. The number of patterns associated with each command is limited due to the small set of system primitives used, and the fact that paraphrase equivalent sentences are mapped into the identical conceptual form. A knowledge-base management and retrieval subsystem [CULL83] provides an efficient method for storing and accessing these patterns.

One of the primary benefits of using this approach is the ability to match against partially formed concepts, or fragments. Since the patterns used are generalized templates, they provide a means for recognizing partially formed concepts due to an incomplete parse. If a command is recognized, the procedures associated with that command will look through the remaining fragments in an attempt to build a correct command.

5.1.1 The Recognition Mechanism

The recognition mechanism is prepared to handle a variety of output from the analyzer, ranging from single, well-formed concepts to a list of fragmented output. There are four conditions the recognizer looks for.

**Condition 1: Does the analyzer output contain a single recognizable result?**
Here the analyzer produces a single, "well-formed" conceptualization of the user's input, which matches against one of the conceptual templates stored with a command package.

**Condition 2: Does the analyzer output contain a single unrecognizable result which needs to undergo a preprocessing stage of recognition?**
In some cases the analyzer output may be unitary, but fail to match against any of the conceptual templates. Ellipses are the most common example of this, and are only "recognized" in context with the previous input.

If the output is fragmented, DESI will check for the following:

**Condition 3: Is there an unknown word amongst the fragments?**
Unknown words are either caused by spelling mistakes, or by words not currently defined in the system. In either case, DESI will try to resolve the meaning of the word by either using a spelling checker, or asking the user if the word could be a synonym for a word already defined in the system.
Having failed all other attempts, DESI will try the following:

**Condition 4: Can DESI recognize any command(s) amongst the fragments?**

If DESI is successful in recognizing a partially formed concept, the meaning of the user's request can be pieced together by using command specific knowledge. Multiclause input may also produce "fragmented" output, in which case each "fragment" is actually a well-formed concept. In this case, DESI recursively runs the recognizer on each fragment, during which the command for each will be determined.

The four conditions given above are designed to catch the important problems that may arise due to user input or analyzer weakness. The bottom-up nature of the analyzer, combined with the recognition and diagnostic phase of processing, provides a very powerful mechanism for robustness.

### 5.2 The Diagnostic Functions

Each command is associated with one or more diagnostic procedures. These procedures are designed to examine the concepts produced by the analyzer to verify that all information needed for successful execution of the command is present. This includes: (1) checking that parameters given are consistent with the database model; (2) filling in parameters not explicitly given; (3) making sure that the functionality requested by the user is consistent with the functionality of the command; and (4) producing a well-formed conceptualization for the command, which will be sent to the generator.

If a user's request is well defined, these processes double-check all information before issuing the MDA command. More commonly these processes make explicit that information which is implicitly given or assumed to be known by the system. For example, DESI must often fill in dimension names as previously shown. DESI also "repackages" the meaning representation provided from the user's request into the form that is needed for the MDA command. For example, user questions are turned into imperative MDA commands. This process not only allows DESI to fill in "default" values not explicitly given, but also allows DESI to double check all of the information provided by the user to make sure it is "legal" and in accordance with the database environment and database model.

Misconceptions about the current state of the environment and/or about the functionality of the
MDA commands causes the user to: (1) exclude information pertinent to the proper execution of a command; or (2) include erroneous information, such as making incorrect associations between items of the database. The diagnostic procedures can detect and resolve these types of errors when they occur.

Errors involving misconceptions about the current state of the database environment can often be directly fixed by DESI. For example, if the user wishes to operate on an MDA object which is not currently in the workspace, DESI will load the object before executing the desired command.

Errors that require interaction with the user involve describing the problem to the user and making available the information needed to correct the error. Normally these errors are caused by misconceptions about the information that an object holds, e.g., specifying dimension values for an object which are not contained in that object; or misconceptions/errors about the information needed to execute a command.

5.2.1 Fragments

Fragmented output from the analyzer has three possible causes: (1) the user uses a word(s) not currently defined in the system's lexicon. (2) the user uses a known word in an unanticipated way, or (3) the fragments are well-formed concepts representing multiclause input.

Natural language interfaces are very susceptible to unknown words [cf., e.g., WEIS85]. The most common cause for unknown words are simple typing mistakes. Normally, a typing mistake can be picked up by a spelling checker. Some typing mistakes, however, result in forming a new word, that may or may not have meaning to the system. In this case it will probably interfere with the NLP mechanism, and either alter the meaning of the user's input, or fail to produce a well-formed meaning representation.

If an unknown word is present amongst the fragments, DESI will: (1) run a spelling checker on it; (2) ask the user if it is a synonym for any of the words known to the system; or (3) ignore the word, and look for an indication of the MDA command amongst the other fragments.

The presence of unknown words that can be resolved provides a means by which new word definitions can be automatically generated "on the fly." Thus, if the user commonly makes the same
spelling error, or has his/her own terminology for other vocabulary, the system will define these words as suited to that particular user.

Missing words and limitations on the word definitions also cause fragmented output. The word definitions cannot be expected to anticipate all "deviant" forms of user input. In the case of missing words, problems arise when the missing word is the main verb which provides the explicit link that ties the sentence together. Despite these limitations, the word definitions will put together as much of the meaning of the input as possible, in the hope that the diagnostician will be able to piece together the overall meaning. In the case that DESI cannot unify the meaning of the fragments into a central concept, it will apologize to the user, giving an explanation of why it could not understand the input.

6.0 Current Status

A prototype version of DESI, programmed in Franz Lisp [FODE81], is currently running under UNIX on a DEC VAX-11/780. In addition to the vocabulary associated with the user objects, the system has a vocabulary of several hundred words, many of them several ways ambiguous. There is also a large phrasal lexicon. For demonstration purposes, DESI communicates with a version of the DSS, ANALECT, running on a PRIME 750, over a dial-up telephone connection. The system has been tested with several simulated marketing databases, a personnel records database, and a real customer database containing over 36,000,000 records. The system's run time is dominated by the analysis module's processing. Currently, the analyzer reads and integrates words from the user's inputs at a rate of approximately 0.5 second/word. Thus, most queries are processed in under 5-6 seconds. Realistically, this is too slow for commercial use when the response time of the DSS is added in.

A major consumer products company, General Foods, Inc., has evaluated the DESI/ANALECT combination, with positive results, to allow its marketing analysis personnel flexible access to the company's large product databases [cf. BUSI84]. A reimplementation of DESI on a single-user workstation for this and similar applications is being planned.
REFERENCES

[ADS80]

[BIER85]

[BIRN81]

[BOBR75]

[BOBR77]

[BOOT85]

[BRAC78]

[BRAC79]

[BROW75]

[BUSI84]

[CARB83]

[CHAR80]

[COM83]

[CULL82]
human factors and user assistance in CAD. Vol. SMC-12, No. 2, pp. 168-182. (Mar/Apr)

[CULL83]

[CULL84]

[CULL86]

[DIAL84]

[DYER84]

[FILM79]

[FODE81]

[HAYE77]

[HAYE83]

[HEND76]

[IP83]

[JAR84]

[KWAS79]
[MAR80]

[MCCA77]

[MDS82]

[NEWE72]

[NORM75]

[RIEG76]

[RIES75]
Riesbeck, C.J. "Conceptual Analysis." In [SCHA75].

[SCHA75]

[STEF80]

[TOIS85]

[WIES85]

[WILK75]

[WINO72]

[WOOD70]

[WOOD75]
Appendix I
BMDA Customization of DESI

The DESI software is stored as a Unix "tar" image on a 1600 bpi, 9-track magtape. After extracting the sources, descend into the dial/nli2 directory, and incant "make diallisp," "make dialliszt," and "make Desi." These will (on a 4.2 or 4.3 BSD Unix system with Franz Lisp Opus 38.91 running on a Vax or Sun3) respectively build a customized version of Lisp and Liszt, and construct Desi itself. The resulting image will be stored as a file "desi" in dial/nli2.

To see what kinds of inputs the system can handle, examine the files in dial/nli2/corpus. The toy BMDA database is stored in dial/nli2i1tm/bmda-db. To run the complete system, type (desi) and then free-form English at the prompt. (For example, "what's in the database?" to see the initial state of the system.) The language-processing software is highly compatible with the materials discussed in Natural Language Processing: A Knowledge-Engineering Approach (indeed, the book amounts to a pedagogical rationalization of this software), and so is easily programmed by anyone familiar with the book's approach.