Multiple Underlying Systems: Translating User Requests into Programs to Produce Answers

Robert J. Bobrow, Philip Resnik, Ralph M. Weischedel

BBN Systems and Technologies Corporation
10 Moulton Street
Cambridge, MA 02138

ABSTRACT

A user may typically need to combine the strengths of more than one system in order to perform a task. In this paper, we describe a component of the Janus natural language interface that translates intensional logic expressions representing the meaning of a request into executable code for each application program, chooses which combination of application systems to use, and designs the transfer of data among them in order to provide an answer. The complete Janus natural language system has been ported to two large command and control decision support aids.

1. Introduction

The norm in the next generation of user environments will be distributed, networked applications. Many problems will be solvable only by use of a combination of applications. If natural language technology is to be applicable in such environments, we must continue to enable the user to talk to computers about his/her problem, not about which application(s) to use.

Most current natural language (NL) systems, whether accepting spoken or typed input, are designed to interface to a single homogeneous underlying system; they have a component geared to producing code for that single class of application systems, such as a single relational database [12]. Providing an English interface to the user's data base, a separate English interface to the same user's planning system, and a third interface to a simulation package, for instance, will neither be attractive nor cost-effective. By contrast, *a seamless, multi-modal, natural language interface will make use of a heterogeneous environment feasible and, if done well, transparent;* this can be accomplished by enabling the user to state information needs without specifying how to decompose those needs into a program calling the various underlying systems required to meet those needs. We believe users who see that NL technology does insulate them from the underlying implementation idiosyncrasies of one application will expect that our models of language and understanding will extend to simultaneous access of several applications.

Consider an example. In DARPA's Fleet Command Center Battle Management Program (FCCBMP), several applications (call them *underlying systems*) are involved, including a relational data base (IDB), two expert systems (CASES and FRESH), and a decision support system (OSGP). The hardware platforms include workstations, conventional time-sharing machines, and parallel mainframes. Suppose the user asks *Which of those submarines has the greatest probability of locating A within 10 hours?* Answering that question involves subproblems from several underlying applications: the display facility, to determine what "those submarines" refers to; FRESH, to calculate how long each submarine would take to get to A's vicinity; CASES, for an intensive, parallelizable numerical calculation estimating the probabilities; and the display facility again, to present the response.

While acoustic and linguistic processing can determine what the user wants, the problem of translating that into an effective program to do what the user wants is a challenging, but solvable problem. In order to deal with multiple underlying systems, not only must our NL interface be able to represent the meaning of the user's request, but it must also be capable of organizing the various application programs at its disposal, choosing which combination of resources to use, and supervising the transfer of data among them. We call this the *multiple underlying systems* (MUS) problem. This paper provides an overview of our approach and results on the MUS problem. The implementation is part of the back end of the Janus natural language interface and is documented in [7].

2. Scope of the Problem

Our view of access to multiple underlying systems is given in Figure 2. As implied in the graphical representation, the user's request, whatever its modality, is translated into an internal representation of the meaning of what the user needs. We initially explored a first-order logic for this purpose; however, in Janus [13] we have adopted an intensional logic [3, 14] to investigate whether intensional logic offers

more appropriate representations for applications more complex than databases, e.g., simulations and other calculations in hypothetical situations. From the statement of what the user needs, we next derive a statement of how to fulfill that need, an *execution plan* composed of abstract commands. The execution plan takes the form of a limited class of data flow graphs for a virtual machine that includes the capabilities of all of the application systems. At the level of that virtual machine, specific commands to specific underlying systems are dispatched, results from those application systems are composed, and decisions are made regarding the appropriate presentation of information to the user. Thus, the multiple underlying systems (MUS) problem is a mapping,

MUS: Semantic representation -- > Program

that is, a mapping from what the user wants to a program to fulfill those needs, using the heterogeneous application programs' functionality.

Though the statement of the problem as phrased above may at first suggest an extremely difficult and long-range program of research in automatic programming (e.g., see [8]), there are several ways one can narrow the scope of the problem to make utility achievable. Restricting the input language, as others have done [4, 6], is certainly one way to narrow the problem to one that is tractable.

In contrast, we allow a richer input language (an intensional logic), but assume that the output is a restricted class of programs: acyclic data flow graphs. The implication of this restriction is that the programs generatable by the MUS component may include only:

- Functions available in the underlying applications systems
- Routines preprogrammed by the application system staff, and
- Operators on those elements, such as functional composition, if-then-else, operators from the relational algebra, and mapping over lists (for instance, for universal quantification and cardinality of sets).

If all the quantifiers are assumed to be restricted to finite sets with a generator function, then the quantifiers can be converted to simple loops over the elements of sets, such as the MAPCAR of Lisp, rather than having to undertake synthesis of arbitrary program loops. We assume that all primitives of the logic have at least one transformation which will rewrite it, potentially in conjunction with other primitives, from the level of the statement of the user's needs to the level of the executable plan. These transformations will have been elicited from the application system experts, e.g., expert system builders, database administrators, and systems programming staff of other application systems. (Some work has been done on automating this process.)

3. Approach

The problem of multiple systems may be decomposed into the following issues, as others have done [4, 9]:

- Representation. It is necessary to represent underlying system capabilities in a uniform way, and to represent the user request in a form independent of any particular underlying system. The input/output constraints for each function of each underlying system must be specified, thus defining the *services* available.
- Formulation. One must choose a combination of underlying system services that satisfies the user request. Where more than one alternative exists, it is preferable to select a solution with low execution costs and low passing of information between systems
- Execution. Actual calls to the underlying systems must be accomplished, information must be passed among the systems as required, and an appropriate response must be generated.

3.1. Representation

3.1.1. Representing the semantics of utterances

Since the meaning of an utterance in Janus is represented as an expression in WML (World Model Language [3]), an intensional logic., the input to the MUS component is in WML. For a sentence such as *Display the destroyers within 500 miles of Vinson*, the WML is as follows:

```
(bring-about
 ((intension
  (exists ?a display
   (object-of ?a
    (iota ?b (power destroyer)
     (exists ?c
      (lambda (?d) interval
       (& (starts-interval ?d VINSON)
        (less-than
              (iota ?e length-measure
                  (interval-length ?d ?e))
              (iota ?f length-measure
                  (& (measure-unit ?f miles)
                    (measure-quantity ?f 500))))))
     (ends-interval ?c ?b))))))
   TIME WORLD))
```

3.1.2. Representing Application Capabilities

To represent the functional capabilities of underlying systems, we define services and servers. A *server* is a functional module typically corresponding to an underlying system or a major part of an underlying system. Each server offers a number of *services*: objects describing a particular piece of functionality provided by a server. Specifying a service in MUS provides the mapping from fragments of logical form to fragments of underlying system code. Each service has associated with it the server it is part of, the input variables, the output variables, the conjuncts computed, and an estimate of the relative cost in applying it.

SAMPLE SERVICES:

```
Land-avoidance-distance:
  owner: Expert System 1
  inputs: (x y)
  locals: (z w)
  pattern:
   ((in-class x vessel)
    (in-class y vessel)
    (in-class z interval)
    (in-class w length-measure)
    (starts-interval z x)
    (ends-interval z y)
    (interval-length z w))
  outputs: (w)
  method: ((route-distance (location-of x)
                           (location-of y))))
  cost: 5

Great-circle-distance:
  owner: Expert System 1
  inputs: (x y)
  locals: (z w)
  pattern:
   ((in-class x vessel)
    (in-class y vessel)
    (in-class z interval)
    (in-class w length-measure)
    (starts-interval z x)
    (ends-interval z y)
    (interval-length z w))
  outputs: (w)
  method: ((gc-distance (location-of x)
                        (location-of y))))
  cost: 1
```

In the example above, there are two competing services for computing distance between two ships: Great-circle-distance, which simply computes a great circle route between two points, and Land-avoidance-distance, which computes the distance of an actual path avoiding land and sticking to shipping lanes. The second is far more accurate when near land, both for calculating delays and in estimating fuel costs; however, the computation time is greater.

3.1.3. Clause Lists

Typically, the applicability of a service is contingent on several facts, and therefore, several propositions must all be true for the service to apply. To facilitate matching the requirements of a given service against the needs expressed in an utterance, we convert expressions in WML to an extended disjunctive normal form (DNF), i.e., a disjunction of conjunctions. We chose DNF because:

- In the simplest case, an expression in disjunctive normal form is simply a conjunction of clauses, a particularly easy logical form to cope with,
- Even when there are disjuncts, each can be individually handled as a conjunction of clauses, and the results then combined together via union, and
- In a disjunctive normal form, the information necessary for a distinct subquery is effectively isolated in one disjunct.

For details of the algorithm for converting an intensional expression to DNF, see [7]; a model-theoretic semantics has been defined for the DNF. For the sentence, *Display the destroyers within 500 miles of Vinson,* whose WML representation was represented earlier, the clause list is as follows:

```
((in-class ?a display)
 (object-of ?a ?b)
 (in-class ?b destroyer)
 (in-class ?c interval)
 (in-class ?d interval)
 (equal ?c ?d)
 (starts-interval ?d VINSON)
 (in-class ?e length-measure)
 (interval-length ?d ?e)
 (in-class ?f length-measure)
 (measure-unit ?f miles)
 (measure-quantity ?f 500)
 (less-than ?e ?f)
 (ends-interval ?c ?b))
```

The normal form in this case is the same as the standard disjunctive normal form: a simple conjunction of clauses. However, there are cases where extensions to disjunctive normal form are used: in particular, certain expressions containing embedded subexpressions (such as universal quantifications, cardinality, and some other set-related operators) are left in place. In such cases, the embedded subexpressions are themselves normalized; the result is a *context* object that compactly represents a necessary logical constraint but has been normalized as far as possible. #S(CONTEXT :OPERATOR FORALL

:OPERATOR-VAR var :CLASS-EXP expression :CONSTRAINT expression) states that *var* is universally quantified over the CLASS-EXP *expression* as var appears in the CONSTRAINT *expression*. As an example, consider the query *Are all the displayed carriers c1?* Its WML expression is given below, followed by its normalized representation.

Note that contexts are defined recursively; thus, arbitrary embeddings of operators are allowed. The component that analyzes the DNF to find underlying application services to carry out the user request calls itself recursively to correctly process DNF expressions invovling embedded expressions.

```
(QUERY
 ((INTENSION
   (PRESENT
    (INTENSION
     (FORALL ?JX699
      (U
       (POWER
        (SET-TO-PRED
         (IOTA ?JX702
          (LAMBDA (?JX701)
           (POWER AIRCRAFT-CARRIER)
           (EXISTS ?JX700 DISPLAY
                   (OBJECT.OF ?JX700 ?JX701)))
          T))))
      (OSGP-ENTITY-OVERALL-READINESS-OF
          ?JX699 C1)))))
  TIME WORLD))

(#S
 (CONTEXT
  :OPERATOR FORALL
  :OPERATOR-VAR ?JX699
  :CLASS-EXP
  ((IN.CLASS ?JX699 AIRCRAFT-CARRIER)
    (IN.CLASS ?JX700 DISPLAY)
    (OBJECT.OF ?JX700 ?JX699))
  :CONSTRAINTS
  ((OSGP-ENTITY-OVERALL-READINESS-OF
      ?JX699 C1))))
```

3.2. Formulation

For a request consisting only of a conjunction of literals, finding a set of appropriate services may be viewed as a kind of set-covering problem. A beam search is used to find a low cost cover. Queries containing embedded subqueries (e.g., the quantifier context in the example above) require recursive calls to this search procedure.

Inherent in the collection of services covering a DNF expression is the data flow that combines the services into a program to fulfill the DNF request. The next step in the formulation process is data flow analysis to extract the data flow graph corresponding to an abstract program fulfilling the request.

In Figure 1, the data flow graph for *Display the destroyers within 500 miles of Vinson* is pictured. Note that the data base (IDB) is called to identify the set of all destroyers, their locations, and the location of Vinson. An expert system is being called to calculate the distance between pairs of locations[1] using land avoidance routes. A Lisp utility for comparing measures is called, followed by the display command in an expert system.

3.3. Execution

In executing the data flow graph, evaluation at a node corresponds to executing the code in the server specified. Function composition corresponds to passing data between systems. Where more than one data flow path enters a node, the natural join over the input lists is computed. Aggregating operations (e.g., computing the cardinality of a set) correspond to a mapping over lists.

4. Challenging Cases

Here we present several well-known challenging classes of problems in translating from logical form to programs.

4.1. Deriving procedures from descriptions.

The challenge is to find a compromise between arbitrary program synthesis and a useful class of program derivation problems. Suppose the user asks for the square root of a value, when the system does not know the meaning of square root, as in *Find the square root of the sum of the squares of the residuals.* Various knowledge acquisition techniques, such as KNACQ [15], would allow a user to provide syntactic and semantic information for the unknown phrase to be defined. Square root could be defined as *a function that computes the number that when multiplied times itself is the same as the input.* However, that is a descriptive definition of square root without any indication of how to compute it. One still must synthesize a program that computes square root; in fact, in early literature on automatic programming and rigorous approaches to developing programs, deriving a program to compute square root was often used as an example problem.

Rather than expecting the system to perform such complex examples of automatic programming, we assume the system need not derive programs for terms that it does not already know. For the example

[1]The distance function takes any physical objects as its arguments and looks up their location.

above, the system should be expected to respond *I don't know how to compute square root.*

By making that assumption, we know that all concepts and relations in the domain model, that is, all primitives appearing in WML as input to the MUS component, have a translation specified by the applications programmer to a composition of underlying services. As stated in Section 2, we further restrict the goals of the MUS component to synthesize programs of a simple structure: acyclic data flow graphs of services where one of the services is applying a function to every element in a finite list. Therefore, the arbitrary program synthesis problem including arbitrary loops and/or recursions is avoided, limiting the scope of inputs handleable but allowing solution of a large class of problems.

To our knowledge, no NL interface allows arbitrary program synthesis. Most assume equivalence at the abstract program level to synthesis of compositions of the select, project, and join operations of relational algebra. Our component goes beyond previous work in that the programs it generates include more than just the relational algebra.

4.2. Side-effects.

It is well-known that generating a program with side-effects is substantially harder than generating a program that is side-effect free. If there are no side effects, transformations of program expressions can be freely applied, preserving the value(s) computed. Nevertheless, side-effects are critical to many interface tasks, for example, changing a display, updating a data base, and setting a value of a variable.

Our component produces acyclic data flow graphs. The only node that can have side-effects is the final node in the graph. This keeps the MUS processing simple, while still allowing for side-effects at the final stage, such as producing output, updating data in the underlying systems, or running an application program having side-effects. All three of those cases have been handled in demonstrations of Janus.

Though this issue has not been discussed in other NL publications to our knowledge, we believe this restriction to be typical in NL systems.

4.3. Collapse of information.

It has long been noted [5] that a complex relation may be represented in a boolean field in a data base, such as the boolean field of the Navy Blue file which for a given vessel was T/F depending on whether there was a doctor onboard the vessel. There was no information about doctors in the data base, except for that field. In a medical data base, a similar phenomenon was noticed [11]; patient records contained a T/F field depending on whether the patient's mother had had melanoma, though there was no other information on the patient's mother or her case of melanoma.

The challenge for such fields is mapping from the many ways that may occur linguistically to the appropriate field without having to write arbitrarily many patterns mapping from logical form to the data base. Just a few examples of the way the melanoma field might be referenced follow:

Did Smith's mother ever have melanoma?
How many patients had a mother suffering from melanoma?
Was melanoma diagnosed for any of the patients' mothers?

Our approach to this problem has been to adopt disjunctive normal form (clause form) as the basis for matching services against requirements in the user request. No matter what the form of user request, transforming it to disjunctive normal form means that the information necessary for a disjunct is effectively isolated in one disjunct. The service represented by the field corresponding to "patient's mother had melanoma" covers two conjoined forms: (MOTHER x y) (HAD-MELANOMA y). All of the examples above, given appropriate definitions of *suffer* and *diagnose*, will have the two relations as conjuncts in the disjunctive normal form for the input, and therefore, will map to the required data base service.

4.4. Hidden joins.

In data bases, a relation in English may require a join to be inferred, given the model in the underlying system. Suppose that a university data base associates an office with every faculty member and a phone number with every office. Additionally, some faculty members may be associated with a lab facility; labs have telphones as well. Then to answer the query, *What is Dr. Ramshaw's phone number?*, the relation between faculty members and phone numbers must be determined. There are two possibilities: the office phone number or the lab phone number.

Most approaches treat this as an inference problem. It can be visualized as finding a relation between two nominal notions *faculty member* and *phone number* [1, 2]. One such path uses the relation OFFICE(PERSON, ROOM) followed by the relation PHONE(ROOM,PHONE-NUMBER). A general heuristic is to use the shortest path. Computing hidden joins complicates the search space in searching for a solution among the underlying services, as can be seen in the architectures proposed, e.g., [1, 4, 9].

In contrast to the typical approach where one

infers the hidden join as needed, we believe such joins are normally anticipatable, and provide support in our lexical definition tools (KNACQ) for specifying them. In KNACQ [15], a knowledge engineer, data base administrator, or other person familiar with the domain and with frame representation specifies for each frame (concept in KL-ONE terminology) and each slot (role in KL-ONE terminology) one or more words denoting that concept or role. In addition, the KNACQ user identifies role chains (sequences of role relations), such as R1(A, B) and R2(B, C), having special linguistic representation. In the example above, KNACQ would prompt the user to select from six possibilities for nominal compounds, possessives, and prepositional connectives relating PERSON to PHONE-NUMBER. In this way, the search space is substantially simplified, since hidden joins have been elicited ahead of time as part of the knowledge acquisition and installation process.

4.5. Data coercion.

At times, the type required by the underlying functions is not directly stated in the input (English) expression but must be derived. One procedure may produce the measure of an angle in degrees, whereas another may require the measure of an angle in radians. Differing application systems may assume a person is referred to by differing attributes, e.g., by social security number in one, but by employee number in another. In *How far is Vinson from Pearl Harbor?*, one must not only infer that the positions of Vinson and Pearl Harbor must be looked up, but also make sure that the coordinates are of the type required by the particular distance function chosen.

In our approach, we assume that there are services available for translating between each mismatch in data type. For the examples above, we assume that there is a translation from degrees to radians and vice versa; that there is a translation from person identified by social security number to person with employee number, and vice versa; that there is a translation function from ships and ports to their location in latitude and longitude. Such translations may already exist in the applications or may be added as a new application. If there are n different ways to identify the same entity (the measure of an angle, a person, the position of a vessel or port, etc.), there need not be (n**2)/2 translation functions of course; a canonical representation may be chosen if as few as 2n translation functions are available to provide inter-translatability to the canonical form.

In constructing the data flow graph, we assume that the canonical representation is used throughout. Then translation functions are inserted on arcs of the data flow graph wherever the output/input assumptions are not met by the canonical form. Of the five challenging problems, this is the only one we have not yet implemented.

5. Related Work

Most previous work applying natural language interfaces provided access to a single system: e.g., a relational data base. Two earlier efforts (at Honeywell [4, 9] and at USC/Information Sciences Institute [6]) dealt with multiple systems. We will focus on comparison with their work.

A limitation common to those two approaches is the minimal expressiveness of the input language: user requests must be expressed as a conjunction of simple relations (literals), equivalent to the select/project/join operations of a relational algebra. This restriction is relaxed in Janus, allowing requests to contain negation of elementary predicates, existential and universal quantification, cardinality and other aggregates, a limited form of disjunction (sufficient for the most common cases), and of course simple conjunction. Wh-questions (who, what, etc.), commands, and yes/no queries are handled, and some classes of helpful responses are produced.

All three efforts employ a search procedure. In the Honeywell effort, graph matching is at the heart of the search; in the USC/ISI effort, the NIKL classifier [10] is at the heart of the search; in our effort, a beam search with a cost function is used.

Only our effort has been tested on applications with a potentially large search space (800 services); the other efforts have thus far been tested on applications with relatively few services.

6. Experience in Applying the System

The MUS component has been applied in the domain of the Fleet Command Center Battle Management Program (FCCBMP), using an internal version of the Integrated Database (IDB) -- a relational database -- as one underlying resource, and a set of LISP functions providing mathematical modeling of a Navy problem as another. The system includes more than 800 services.

An earlier version of the system described here was also applied to provide natural language access to data in Intellicorp's KEE knowledge-base system, to objects representing hypothetical world-states in an object-oriented simulation system, and to LISP functions capable of manipulating this data.

We have begun integrating the MUS component with BBN's Spoken Language System HARC.

7. Conclusions

The work offers highly desirable utility along the following two dimensions:

- It frees the user from having to identify for each term (word) pieces of program that would carry out their meaning.
- It improves the modularity of the interface, insulating the presentation of information, such as table i/o, from details of the underlying application(s).

We have found the general approach depicted in Figure 2 quite flexible. The approach was developed in work on natural language processing; however, it seems to be valuable for other types of I/O modalities. Some preliminary work has suggested its utility for table input and output in managing data base update, data base retrieval, and a directly manipulable image of tabular data. Our prototype module generates code from forms in the intensional logic; then the components originally developed for the natural language processor provide the translation mechanism to and from intensional logic and underlying systems that actually store the data.

Acknowledgments

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by ONR under Contracts N00014-85-C-0079 and N00014-85-C-0016. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

The current address for Philip Resnik is Computer & Information Sciences Department, University of Pennsylvania, Philadelphia, PA 19104.

We gratefully acknowledge the comments and assistance of Lance Ramshaw in drafts of this paper.

REFERENCES

1. Carberry, M.S. Using Inferred Knowledge to Understand Pragmatically Ill-Formed Queries. In R. Reilly, Ed., *Communication Failure in Dialogue*, North-Holland, 1987.

2. Chang, C.L. Finding missing joins for incomplete in Relational Data Bases. Research Report RJ2145, IBM Research Laboratory, 1978. San Jose, CA.

3. Hinrichs, E.W., Ayuso, D.M., and Scha, R. The Syntax and Semantics of the JANUS Semantic Interpretation Language. In *Research and Development in Natural Language Understanding as Part of the Strategic Computing Program, Annual Technical Report December 1985 - December 1986*, BBN Laboratories, Report No. 6522, 1987, pp. 27-31.

4. Kaemmerer, W. and Larson, J. A graph-oriented knowledge representation and unification technique for automatically selecting and invoking software functions. Proceedings AAAI-86 Fifth National Conference on Artificial Intelligence, American Association for Artificial Intelligence, 1986, pp. 825-830.

5. Moore, R.C. Natural Language Access to Databases - Theoretical/Technical Issues. Proceedings of the 20th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, June, 1982, pp. 44-45.

6. Pavlin, J. and Bates, R. SIMS: single interface to multiple systems. Tech. Rept. ISI/RR-88-200, University of Southern California Information Sciences Institute, February, 1988.

7. Resnik, P. Access to Multiple Underlying Systems in Janus. BBN Report 7142, Bolt Beranek and Newman Inc., September, 1989.

8. Rich, C. and Waters, R.C. Automatic Programming: Myths and Prospects.

9. Ryan, K. R. and Larson, J. A. . The use of E-R Data Models in Capability Schemas. In Spaccapietra, S., Ed., *Entity-Relationship Approach*, Elsevier Science Publishers, 1987.

10. Schmolze, J.G., Lipkis, T.A. Classification in the KL-ONE Knowledge Representation System. Proceedings of the Eighth International Joint Conference on Artificial Intelligence, 1983.

11. Stallard, D.G. A Terminological Simplification Transformation for Natural Language Question-Answering Systems. Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics, New York, June, 1986, pp. 241-246.

12. Stallard, David. Answering Questions Posed in an Intensional Logic: A Multilevel Semantics Approach. In *Research and Development in Natural Language Understanding as Part of the Strategic Computing Program*, R. Weischedel, D.Ayuso, A. Haas, E. Hinrichs, R. Scha, V. Shaked, D. Stallard, Eds., BBN Laboratories, Cambridge, Mass., 1987, ch. 4, pp. 35-47. Report No. 6522.

13. Weischedel, R., Ayuso, D., Haas, A., Hinrichs, E., Scha. R., Shaked, V., Stallard, D. Research and Development in Natural Language Understanding as Part of the Strategic Computing Program. BBN

Laboratories, Cambridge, Mass., 1987. Report No. 6522.

14. Weischedel, R. M. A Hybrid Approach to Representation in the Janus Natural Language Processor. Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics, 1989, pp. 193-202.

15. Weischedel, R.M., Bobrow, R., Ayuso, D.M., and Ramshaw, L. Portability in the Janus Natural Language Interface. Speech and Natural Language, San Mateo, CA, 1989, pp. 112-117.

SENTENCE:
"Display the destroyers within 500 miles of Vinson."

DATA FLOW GRAPH:

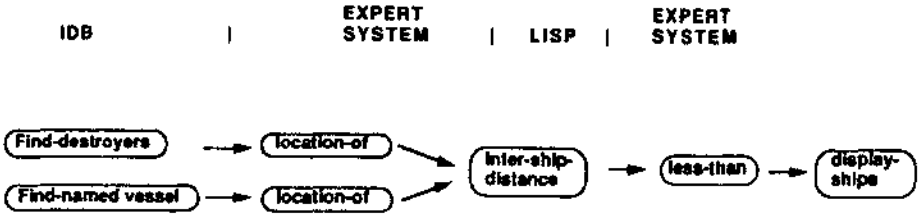

Figure 1: Data Flow Graph for "Display the destroyers within 500 miles of Vinson"

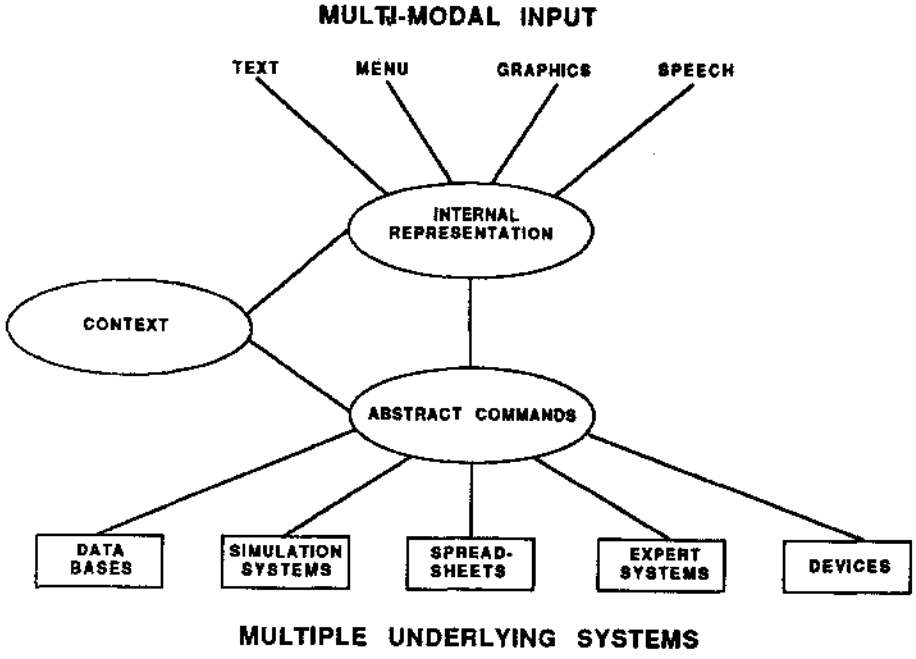

Figure 2: BBN's Approach to Simultaneous Access to Multiple Systems