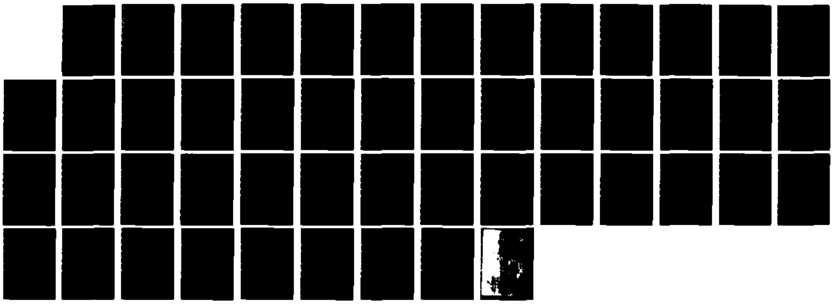
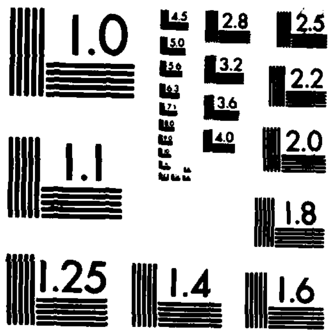
AD-A127 739 AN EXPERT DISTRIBUTED ROBOTICS SYSTEM WITH COMPREHENSION AND LEARNING ABI..(U) ILLINOIS UNIV AT URBANA COORDINATED SCIENCE LAB D L WALTZ ET AL. FEB 83 1/1

UNCLASSIFIED T-123 AFOSR-TR-83-0334 F49620-82-K-0009 F/G 6/4 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AFOSR-TR- 83-0334

(3

REPORT T-123 FEBRUARY, 1983

COORDINATED SCIENCE LABORATORY

F49620-82-K-0009

AD A127739

AN EXPERT DISTRIBUTED ROBOTICS SYSTEM WITH COMPREHENSION AND LEARNING ABILITIES IN THE AIRCRAFT FLIGHT DOMAIN

D.L. WALTZ
G. DEJONG
R.T. CHIEN

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

83 05 06-105

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 83-0334	2. GOVT ACCESSION NO. AD-A127 739	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) AN EXPERT DISTRIBUTED ROBOTICS SYSTEM WITH COMPREHENSION AND LEARNING ABILITIES IN THE AIRCRAFT FLIGHT DOMAIN		5. TYPE OF REPORT & PERIOD COVERED INTERIM, 1 JAN 82-31 DEC 82
		6. PERFORMING ORG. REPORT NUMBER T-123
7. AUTHOR(s) D.L. Waltz, G. DeJong and R.T. Chien		8. CONTRACT OR GRANT NUMBER(s) F49620-82-K-0009
9. PERFORMING ORGANIZATION NAME AND ADDRESS Coordinated Science Laboratory University of Illinois at Urbana-Champaign Champaign IL 61820		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS PE61102F; 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS Mathematical & Information Sciences Directorate Air Force Office of Scientific Research Bolling AFB DC 20332		12. REPORT DATE FEB 83
		13. NUMBER OF PAGES 44
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION ST. 4ENT (of t e abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY TES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		

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

The goals of this project are to better understand the processes involved in reasoning about and troubleshooting physical mechanisms. This will lay the foundation for the design of expert systems to carry out these processes automatically. Of particular interest are results that will allow novel applications for the aircraft flight domain. The work falls into three broad areas: (1) understanding natural language; (2) learning; and (3) mechanism modeling.

Understanding Natural Language: By coupling a natural language (NL) (CONTINUED)

DD FORM 1 JAN 73 1473

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ITEM #20, CONTINUED: interface with sensing, reasoning and learning programs, a range of systems may be produced with desirable properties not possible with today's technology. NL interfaces are well suited for tasks where a computer system is to be directed by a computer-naive user; for tasks which computer-knowledgeable users do infrequently; in situations where English is more concise than formal languages; and for tasks where system knowledge is expressed in English (e.g., an on-board aircraft system which uses text from manuals to guide troubleshooting, repair, or error-recovery procedures).

Learning: The investigators would like their systems to improve in performance as time goes by, without having to be completely reprogrammed. Thus, it is important to devise system designs that can support incremental improvement. Researchers are investigating both problems of learning by being told, and problems of learning from experience. In both forms of learning, they want their systems to generalize their new knowledge, and not simply remember literally the situations they encounter or are told about.

Mechanism Modeling: Researchers want their systems to use English and learn about the domain of the operation and structure of physical mechanisms. They are concerned here with the representation of systems, subsystems, and components, with normal operation, failures, and with the processes of troubleshooting, repair, and error-correction. Such mechanism modeling depends on building powerful representation facilities for causality, time, and spatial arrangement.

REPORT T-123

Annual Technical Report for Research in

AN EXPERT DISTRIBUTED ROBOTICS SYSTEM WITH COMPREHENSION AND LEARNING ABILITIES IN THE AIRCRAFT FLIGHT DOMAIN

For the Period

January 1, 1982 - December 30, 1982

Submitted to

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH

Investigators:

D. L. Waltz, G. DeJong, R.T. Chien

February 1983

TABLE OF CONTENTS

1. Research Objectives

The goals of this project are to better understand the processes involved in reasoning about and troubleshooting physical mechanisms. This will lay the foundation for the design of expert systems to carry out these processes automatically. Of particular interest are results that will allow novel applications for the aircraft flight domain. ~~Our~~ The work falls into three broad areas:

~~(1)~~ understanding natural language;
~~(2)~~ learning; and
~~(3)~~ mechanism modeling.

1.1. Understanding Natural Language

By coupling a natural language (NL) interface with sensing, reasoning and learning programs, a range of systems may be produced with desirable properties not possible with today's technology. NL interfaces are well suited for tasks where a computer system is to be directed by a computer-naive user; for tasks which computer-knowledgeable users do infrequently; in situations where English is more concise than formal languages; and for tasks where system knowledge is expressed in English (e.g., an on-board aircraft system which uses text from manuals to guide troubleshooting, repair, or error-recovery procedures).

1.2. Learning

We would like our systems to improve in performance as time goes by, without having to be completely reprogrammed. Thus it is important to devise system designs that can support incremental improvement. We are investigating both problems of learning by being told, and problems of learning from experience. In both forms of learning, we want our systems to generalize their new knowledge, and not simply remember literally the situations they encounter or are told about.

1.3. Mechanism Modelling

We want our systems to use English and learn about the domain of the operation and structure of physical mechanisms. We are concerned here with the representation of systems, subsystems, and components, with normal operation, failures, and with the processes of troubleshooting, repair, and error-correction. Such mechanism modeling depends on building powerful representation facilities for causality, time, and spatial arrangement.

2. Representations and Natural Language Processing

Professor David Waltz

David T. Spoor, Graduate Student

Raman Rajagopalan, Graduate Student

Any system which is to operate in the aircraft flight domain (or any other such real-world domain) must have sufficiently solid representational underpinnings upon which to build its model of understanding. Such underpinnings must include not only a method for representing aircraft related objects and their functional interrelationships, but must also include a method for representing (and therefore modeling) these relationships as they change in time.

Since the system must represent time as well as time-varying relationships, it is desirable that the system do so in a manner which is closely compatible with the way that humans represent these same concepts. This compatibility is not only desirable but necessary if humans are to easily understand the system's decision-making, or maintain the system.

Over the reporting period, one of the goals of this research project has been the formalization of a representation for time-varying objects and relationships based on a humanly comprehensible model. At the same time, work has been directed to the parallel problem of the representation of knowledge in the aircraft engine domain.

2.1. Temporal Representations

Temporal knowledge plays a fundamental role in not only our understanding of times and dates, but in our understanding of basic natural language as well as our planning and remembering processes. The goal of this research is to design and construct a natural language system which will extract temporal knowledge from language, as well as construct inferences which are commonly made from that knowledge, and link these to knowledge about causality and space. The design of the system is based on not only past work in temporal and spatial representation, but on the ongoing research in natural language universals being conducted by La Raw Maran here at CSL.

The design of the system is well underway with the specification of the low level time interval modeling system based on universal natural language categories almost complete. This work will continue as spatial and causal modeling subsystems are added to the design. The implementation of the system in Interlisp [Teitelman78] will be aided by utilizing the RUS parser [Mark80] to process raw natural language data, and the forthcoming New Implementation of KL-ONE (NIKL) [Schmolze82] for knowledge representation, both products of BBN. Current goals include both a working version of the RUS parser as well as an implementation of the low-level time interval subsystem within the next few months, and a system which can model and answer questions about simple temporal relationships within a year.

2.1.1. Natural Language Temporal Inference and Knowledge (NALATIK) System

A temporal representation system is a partial ordering of "temporary things" (things that take time), but what are these "things"? By looking to natural language we can come up with a taxonomy of "things" that take time. By analyzing a number of widely differing languages we have developed a core set of requirements for a natural language temporal knowledge representation and reasoning system [Maran83]. By utilizing our own linguistic data, and learning from the past work of McDermott [McDermott82] in temporal logic, and the work of Allen [Allen81] and Vilain [Vilain82] in temporal representation, we are developing a temporal representation and inference system capable of understanding the time content of English sentences. This system, while developed for the English language should not be restricted to English, since it is based on natural language cognitive universals, and should be readily adaptable to other natural languages. We believe that through careful analysis of many languages the conceptual temporal model which underlies language can not only be discerned, but that this temporal model underlies all of temporal understanding, not just language. Our language understanding system is based on this universal temporal model.

Once our core temporal representation is developed we shall prove its salience by applying it to simple applications involving raw natural language input, and analysis in a restricted domain world. This step will involve utilization of a parsing system for English, as well as development of a general world knowledge system to enable event comprehension. Current plans involve use of the RUSgrammar parsing system [Mark80] for parsing English input and New Implementation of KL/ONE (NIKL) as a knowledge representation system. Both RUS and NIKL are currently ongoing research projects at BBN. By using already developed parsing and knowledge representation systems we hope to significantly shorten the development time of our system. In the end the system's capacity to understand language will be demonstrated by its ability to respond to questions posed in English, about its temporal knowledge. It is also our hope that ideas developed by this project will prove useful in the mechanism modeling and robotics projects which constitute the rest of this project.

While the current major thrust of our research is development of a temporal representation for language, issues of spatial representation as well as causality must be included in any useful system, since many events which take time involve changes in physical configuration. Other considerations involve understanding of plans and goals as potential future events, as well as the planning process as preparation for future events. It is our conviction that these issues are intimately linked to any temporal knowledge system, and must be considered early in our design process.

2.1.2. Current Status

While most of the work conducted so far has been in development of the core temporal model, some progress has been made in implementation. Interlisp-VAX is our language of choice for initial implementation. This choice is based on the ability to run identical code on VAX as well as on Xerox D machines, the availability of the RUS parser as well as KL/ONE and NIKL (soon) in Interlisp, and of course the overall highly mature Interlisp program development environment. Within the past few weeks initial development of a first generation core temporal model has been undertaken. A copy of the recently revised RUS parser has been obtained from BBN and is currently undergoing conversion to Interlisp-VAX from Interlisp-10, as well as adaptation to our own lexicon. We hope within the next few weeks to complete implementation of the core temporal model and to move it quickly into an Interlisp implementation. Within the next few months we hope to have the RUS parser operational as well as begin connecting the parser to our modeling system. Also we hope to obtain and install the second release of Interlisp-VAX, which promises to be much faster than the current implementation.

2.1.3. Psychological Foundations

The categories and classification system proposed herein are the result of the study of all the languages surveyed. The most basic role of time lies in understanding the meaning of events in the real world.

We understand events in terms of their duration, serial order of occurrence, overlappings, the aspects of beginning and concluding, their decomposition into component events, and so forth. States, the complementary elements of events, are understood in a like manner, but lack durational information. The most basic elements of our system are states and events. This distinction corresponds to the differentiation between process and state verbs in natural language. Events are processes which have a "shape" as described by an event shape diagram (see [Waltz82a]), and as a consequence are bounded in time. States on the other hand have no "shape", remaining constant through time, and are bounded by events which create and destroy them. States in themselves have no inherent duration, but must rely on their bounding events for durational definition.

Events and states can be further classified into retrospective, present, and modal categories. Retrospective events are events which have happened in real time and have finished. These are processes which have occurred and are completed or states which are bounded on both ends by events which have occurred or are occurring. For example the sentence:

The test yesterday was successful.

is an example of a retrospective event, whereas the sentence:

John was in Washington last week.

is an example of a retrospective state. Present events are processes which have started, but not completed, and states which have their onset boundary defined, by retrospective event, and completion boundary not well defined. The sentences

The test is going well.

John is in Washington.

are examples of present events. Modal events are events which might have happened in the past, might be occurring presently, or may occur in the future. For example:

The test is scheduled for next week.

John could have been in Washington now.

These categories in language are determined from the tense and aspect of the verb used.

Durational information in our system is represented by a combination of time intervals and points in time. By allowing for both intervals and points we have the ability to simplify durational information by converting an interval to a point in time when it is not relevant to the representation being constructed. This feature is reflected in language by the ability to use both punctual and durative verbs. For example the sentence:

Just before the end of the test the light flashed.

describes both a time interval, the interval of the test, and a point in time, the instant that the light flashed. In the context of this example sentence, the light's flashing was instantaneous, however we also allow for this point in time to be interpreted as a time interval in another context. If the next sentence was:

The light went on because of a dip in the pressure of the main fuel system.

the light's lighting would cover an interval of time, which matched the process of the fuel pressure falling and rising. We also allow for events to have substructure which expands on their mechanism. For example:

The light flashed during the test.

This event is described as an interval of time in which an unknown number of repetitions of the event "flash" happen. The event "flash" in turn is described as the turning on and off of the lamp. We have a similar substructure to account for the measuring of time through regular events such as days, hours, and seconds. Other event substructures are composed by describing the events process in similar ways.

Relations between intervals in our system are defined by linking intervals which are used to denote before and after relations. This allows us to assign duration to the before and after links and thus capture more accurately the separation of two events. Our system has no representation for "now", but rather relies on marking of individual events as potential, occurring, or past events.

Causality and its relation to processes and states is a subject which we have only begun to explore, and we hope to be able to present similar results for causality in the future.

2.2. Knowledge Representation in the Engine Domain

As of the last progress report [Waltz82b], work was beginning on the representation of an aircraft gas turbine engine [Group80] for a database query-response system. This system is to be capable of answering questions based on its internal model of the gas turbine engine through a reasoning process, as opposed to a simplistic "canned" response database query system. As an example of the type of questioning this system is expected to handle, consider the following:

(1) The user tells the system that:

(a) the engine is making abnormal noises,
(b) the airplane is on the ground,
(c) there is a flame at the engine exhaust.

(2) The system responds that one possible cause of these symptoms is "compressor stall."

(3) The user inquires what reasons there might be for compressor stall.

(4) The system might reply that the stall is due to high crosswinds causing distortion of input to the compressor. In order to make this reply, however, the system must have access to information

regarding the existence of high crosswinds. If the system does not posses direct access to this information, the system should ask the user whether there are indeed high crosswinds.

The reasoning process itself has been dealt with by other artificial intelligence researchers, especially [Doyle78] and [Stallman79]. Other related work will be found in [Forbus80] and [Rieger75]. Most of the work in this portion of the project during this period has been to become more knowledgeable about the operation and troubleshooting of gas turbine engines, and about the problem-solving and representational areas of AI that are relevant to this problem.

3. Explanatory Schema Acquisition

Professor Gerald DeJong

Christian Debrunner, Graduate Student

Paul Harrington, Graduate Student

Alberto Segre, Graduate Student

Large amounts of knowledge are an essential component of any system that is to demonstrate intelligent behavior. Without a certain amount of background knowledge about the problem domain in which the problem solver is to operate, the system cannot be expected to make reasonable decisions. The presence of such a large knowledge base requires an efficient method of indexing and recovering information during the problem solving activity. One method of organizing knowledge in artificial intelligence systems is the schema construct. A schema can be considered to be a "chunk" of related information.

The use of schemata as a "knowledge handling" mechanism makes no assumption about the methods used to acquire this knowledge. In fact, most artificial intelligence researchers explicitly endow their systems with the requisite background knowledge. This is hardly ideal, since this process of hand-coding schemata is often both difficult as well as time-consuming. A better solution would be for the system to acquire its own background knowledge in the course of solving simple domain specific problems. This schema construction process has as its human equivalent the learning process.

This part of the project is concerned with knowledge acquisition, or more precisely, schema acquisition.

3.1. Learning in the Robot Domain - Robot Schema Acquisition

Most robots seeing industrial use today have practically no intelligence. They are programmed to repeat certain motions over and over, for instance, to move the hand to a certain position and close the grippers, grasping a bolt at that position. This sequence of actions is programmed explicitly, usually by having the human programmer physically move the robot arm and hand into the desired positions and having the controlling computer store the motion sequence. If the motion sequence is very long or if it must be modified very often, programming in this manner can become very tedious.

More useful would be a robot system that could learn a task in a manner similar to how a human might learn it. For instance, if the robot system is given some idea of purpose, some idea of what the goal of going through the given motions is, it can use the sequence of motions (or a slight modification of the sequence of motions) in another situation where the same goal arises. With this kind of knowledge, if the robot system is given a sequence of instructions it can analyze them and learn a new command from them by reasoning about how the goals of the individual motions combine to achieve the overall goal of the whole sequence of instructions.

The robot learning system now under development will implement these ideas. The learning techniques it will use are outlined in [Waltz82b].

Development of Interlisp-VAX code for the preliminary version of the system is proceeding along two fronts:

(1) Coding of a supervisory program which, in this preliminary version, contains only very simple schemata, e.g. schemata for moving the hand, for opening and closing the grippers and for rotating the wrist joint.

(2) Coding of a robot control language to allow the supervisory program to send very simple commands to the robot which will move the hand parallel to the axes of a rectangular coordinate system. The simplicity of movements restricted to six directions reduces the complexity that the robot system encounters in planning out its motions and should permit thorough checking of the correctness of the methods being used.

3.1.1. The Robot Supervisor

This part of the project is oriented towards the production of a robot supervisory program which will eventually be capable of using the learning techniques outlined in [DeJong82a]. The first of these (and the one which the first version of the system will implement) is called "schema composition". A schema in this first version system is considered to be a unit of information about a particular action, e.g. what conditions must exist before the action can be performed, how to perform the action, and what the results of performing the action will be. Given schemata about various actions, the program can reason about a sequence of these actions, and can derive a new schema containing the same sorts of information about this sequence. This is called schema composition because it involves composing (connecting) two or more schemata to make a larger one. The other three techniques outlined in [DeJong82b] are "secondary effect elevation," "schema alteration," and "volitionalization." The preliminary version of our system is implementing schema composition. We may later extend the system to encompass schema alteration as well.

The robot's first task is to learn how to "PICKUP" a block. Since it has no information about "picking up", we have to show it how by giving it the motions to go through to PICKUP a particular block. From this sequence of actions that we give it, the robot program builds a new schema for PICKUP with all the proper preconditions actions and results. In a similar manner we will show the system other operations, building towards increasingly complicated functions.

It is possible that the schemata the system builds up in this manner will not always work quite properly. For this reason, it is desirable for the the system to be able to execute "schema alteration" type learning to correct the problem. In order to do this, the system requires relatively sophisticated knowledge about what can go wrong, and what kinds of things to change when things do go wrong. Schema alteration is beyond the capability of the preliminary version of the system.

Eventually we hope to augment the system with visual feedback. In this first version, the system must be told where all the objects in its world are, and if it drops a block or knocks over some blocks, in most cases the robot program would have to be informed externally. The addition of computer vision would allow the system to notice this type of unexpected event and perhaps take some corrective action. Visual feedback is still a future addition, however, and is not terribly important for this first version.

3.1.2. The Implicit Robot Control Language

In the present industrial environment robots are programmed explicitly; robot control programs explicitly state the position and orientation that each joint of the robot must have for the successful completion of an assigned task. Implicit Robot Control uses Artificial Intelligence concepts to create a sense of purpose for the robot, allowing higher-level programming for the robot user while simultaneously avoiding common errors associated with explicit programming.

An example of a common error with explicit programming is a robot trying to PICKUP a bolt at a certain location regardless of whether the bolt is actually there or not. The robot has learned to PICKUP using a set sequence of actions such as:

(1) move to a certain (fixed) location;
(2) squeeze the gripper;
(3) lift upward.

Notice that in this sequence the bolt is never mentioned; the robot is just as happy to PICKUP nothing as it is to PICKUP a bolt.

Implicit Programming is concerned with making sure the robot does, in fact, PICKUP the bolt, not just execute a sequence of predetermined actions. Implicit Programming also allows the robot to generalize the actions it must execute in PICKUP (or any other task) so as to be able to PICKUP a bolt anywhere within its reach, even though the robot may have been shown PICKUP only once, at a specific location.

Work during the reporting period was aimed at applying Implicit Programming to a real robot, a Stanford Manipulator. The Implicit

Programs will not directly manipulate the Stanford Robot Arm, instead they will manipulate a computer-modeled, hypothetical robot arm. The Implicit Programs will send command signals to the hypothetical robot arm to control its imaginary movement. This part of the project concentrates on using these commands as the input for a set of Pascal programs that will in turn control the simultaneous movement of all seven joints of the Stanford Manipulator.

Initially the hypothetical arm will move in only one cartesian direction at a time, while leaving the other two coordinates unchanged. The Pascal programs control the command signals for each joint of the Stanford Arm so that the resulting motion will be in a straight line in only one direction, allowing the Stanford Arm to duplicate the movement of the hypothetical arm. In addition to straight line motion control, these Pascal programs maintain the initial orientation of the Stanford Robot Arm's gripper throughout the entire movement. If the Implicit Programs command a change in the hypothetical arm's gripper orientation, then the Pascal programs must also re-orient the Stanford Arm's gripper a corresponding amount. Also, when the hypothetical arm must close its gripper fingers), then the Pascal programs must command the Stanford Arm to close its gripper as well.

After the Stanford Arm has completed the actions required to model the actions completed by the hypothetical arm, the Pascal programs will report back to the Implicit Programs that the requested task has been completed. This will allow the Implicit Programs to update the internal world model that the hypothetical robot arm exists in. If the Stanford Arm cannot complete a required task because of physical limitations (the

Stanford arm might not be able to rotate a certain joint to its desired position because that joint may already be at maximum rotation), the Pascal programs will return the Stanford Arm to the position it had at the beginning of the task in question, while sending an appropriate error message to the Implicit Programs.

3.2. Learning in the Story Processing Domain

Schema-based systems have shown much promise in the quest for the construction of a natural-language understanding system [DeJong82b]. A schema is a collection of objects, events and actions which are packaged together to provide a natural-language understanding system with enough background knowledge to make sense of its input. For example, if the system were given the input "John ate his lunch out of a can," the system would require some sort of background knowledge about food packaging (i.e., that food may be packaged in tin cans) in order to understand this particular input. This kind of background information (normally referred to as "world knowledge") may be contained in a schema [DeJong79].

Providing such a system with enough world knowledge in the form of schemata is unfortunately not an easy task. The process of hand-ccding schemata is tedious at best, and there is no certainty that a schema so produced is complete and error-free. Therefore, it would be desirable for the system to acquire its own schemata with some sort of learning process: this would allow automatic construction of new schemata simply by giving the system new inputs to analyze. An example of this type of process is called "explanatory schema acquisition" and is outlined in [DeJong82a].

During the reporting period work has focused on an implementation of such a schema acquisition system in the story-processing domain. The idea is that given a story input for which the system has no matching schema the system should be capable of either creating a new schema, modifying an existing schema, or combining several existing schemata in order to provide enough world knowledge to adequately explain (and therefore "understand") the story input. This implementation should help to flesh out the ideas expressed in [DeJong82a], providing a testing ground on which to check these ideas for completeness and consistency.

3.2.1. Progress to Date

The first step in constructing a system as described in the previous section is to determine the exact format of the more important data structures (i.e., what exactly comprises a schema) as well as the procedure the system is to follow.

The input to the system will be a formalized representation of the story, just as it would appear after the initial parsing process. The system must take these story inputs and create some sort of conceptual representation or model of the story as a whole, inferring missing but causally necessary events.

The representation adopted for this story model is in a graph paradigm and therefore consists of a collection of nodes and (directed) links [Bondy76]. A node may represent either:

(1) an object,
(2) a prototypical object or object category,
(3) an event or action, or
(4) a relationship between objects.

Each object represented in the model is represented by one and only one node in the graph: whether it be a physical object (such as a shoe), a particular living object, or an abstract object (such as last night's headache). This object may have some _specific_ attributes or properties associated to it. For example, if the story input mentions Fred, the grocery clerk, the model of this object must reflect that this object's name is Fred and that this object's occupation is grocery clerk. In addition, this object inherits some additional _general_ attributes or properties by virtue of its membership in a larger class of prototypical objects. In our example, the node representing the Fred object belongs to that class of prototype objects called "person" and therefore shares all those default attributes (i.e., those inheritable properties such as "has two legs" in the case of a person) and constraints (i.e., those constraints on an item's possible values such as "less than 8 feet" for the "height" attribute of objects belonging to the "person" prototype) normally attributed to "persons". This inheritance mechanism is typical of frame-based knowledge representation systems. Therefore, the following distinction could be made: world knowledge about actions or events is encoded in schemata while world knowledge about objects is encoded in these prototype objects (similar to the frame concept first introduced by Minsky: see [Minsky75]).

3.2.2. _The Next Step_

Future work will continue with the implementation of this schema acquisition system. Interlisp-VAX code which will make the input-to-graph mapping is currently being developed. Once this code is complete, work can begin on the implementation of the explanatory schema acquisition constructs within the framework of this graph model. The graph model provides a basis in graph theory which will be useful, for example in checking stories for similarities (isomorphisim in the graph domain).

3.2.3. Sample Input

What follows is a sample input story. The story deals with a hypothetical kidnapping, for which the system does not already have a schema. Those entries beginning with an asterisk correspond to the original English input, and are disregarded by the model builder (they are included only for clarity).

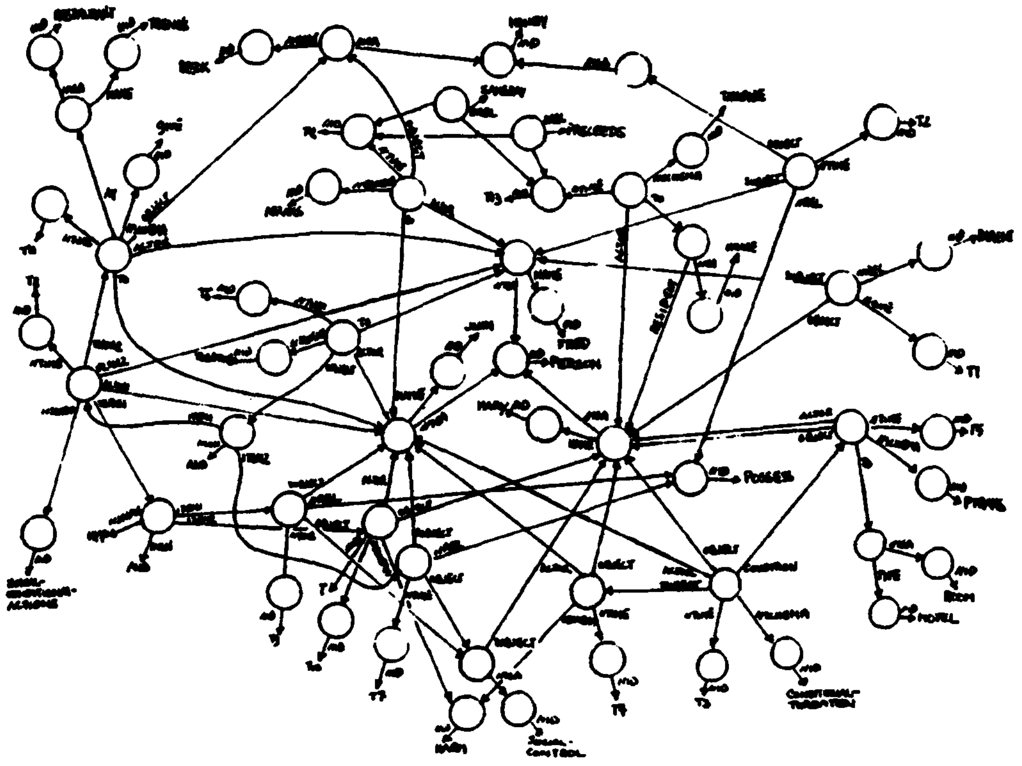
```
(* Mary is Fred's daughter.)

(parent (subject (fred)) (object (mary)) (time t1))

(* Fred is rich.)

(possess (subject (fred)) (object (money)) (time t2))

(* John coerced Mary into his motel room.)

(conditional-threaten (actor (john)) (subject (mary)) (time t3)
    (threat (harm (actor (john)) (object (mary)) (time t4)))
    (condition (ptrans (actor (mary))
                       (object (mary))
                       (to (room (type (motel))
                                 (resident (john)))) (time t5))))

(* John called Fred and said that he had Mary and that he would not
 harm her and he would release her if Fred delivered $250k to John at Treno's
 restaurant.)

(telephone (time t6)
           (actor (john))
           (to (fred))
           (object (and
                     (posses (subject (john))
                             (object (social-control
                                         (subject (mary)))) (time t7))
                     (mutual-conditional-actions
                          (actor1 (john))
```

```
                                            (terms1 (status hyp)
                                                (and
                                                 (not (possess (actor (john))
                                                          (object (social-control
                                                                      (subject (mary))))
                                                          (time t9) ))
                                                 (not (harm (actor (john))
                                                            (object (mary)) (time t10)))))
                                            (actor2 (fred))
                                            (terms2 (status hyp)
                                                 (give (actor (fred))
                                                       (to (john))
                                                       (object (money (amount ($250k))))
                                                       (at (restaurant (name (trenos))))
                                                       (time t11)))
                                            (time t8)))))

(* Fred gave John the money.)

(atrans (time t12) (actor (fred)) (to (john)) (object ($250k)))

(* Later that day, Mary arrived home in a taxi.)

(taxiride  (time t13) (actor (mary)) (to (house (resident (mary)))))

(sameday (time t12) (time t13))
(precedes (time t12) (time t13))
```

3.2.4. Graph Representation of Sample Input

A sketch of the interrelationships expressed in the above input is included at the end of this section of the report. Nodes with //ID links are objects or prototype objects, while nodes with //REL or //SCHEMA links correspond to references to schemata or relations, respectively.

An interesting and useful result of this representation is that schemata are inherently hierarchical: that is they correspond to sets of interconnected schemata and relations. In addition, this type of representation is very well suited to implementation in LISP. A node and link representation allows information about a particular object in a story to be concentrated in a single location. Localizing references to an object permits inheritance of attributes from prototype objects across //ISA links. Such inferences, which are often vitally important to understanding a story, add yet more information to nodes.



Graph Representation of Sample Input

4. Diagnosis and Design of Mechanisms

Professor R. T. Chien

William Frederick, Graduate Student

Michael Houghton, Graduate Student

Adam Pajerski, Graduate Student

This section of the project deals with the development of computer-based diagnosis and design methodologies rooted in deep-level understanding models. Current work is aimed at producing representations in the aircraft domain which permit investigation of diagnosis in multiple fault situations.

4.1. Knowledge Representation and Functional Dependencies

During the reporting period, work began on a method which would allow the computer to understand aircraft systems by applying engineering knowledge to a technical manual diagram. This methodology is needed by the distributed robotics system to perform fault diagnosis as well as to enter new device descriptions into the system. Current work is aimed at producing a model which can represent mechanisms in the computer. We intend to make this model similar to the mental model that humans form when attempting to understand engineering systems.

As a first step in checking the computer's understanding of an engineering system, a "common sense algorithm" (CSA) like diagram [Rieger common sense] for the system will be produced from the computer understanding model. This in itself presents a problem, since the CSA representation of a mechanism is not unique. More uniformity in the CSA representation of a mechanism seems to be possible if the CSA representation is drawn at a specific level of detail.

Using the electrical generating system of a DC-10 as an example, three levels of detail were considered: the _system level_, the _subsystem level_, and the _component level_. The system level and the component level are the most clearly defined because they represent the top and bottom levels (or most and least detailed), respectively. The subsystem level uses a rather arbitrary division of the main system which may cause some ambiguity. More complex systems may require a fourth level between the subsystem level and the component level.

Each part of the human understanding process must be represented in the computer in order to produce a similar understanding processes. Similarity between computer and human modeling processes is desirable for reasons of maintainability and understandability of the system, as well as for providing easily understandable explanations of the system's behavior.

The inputs for the human understanding process are an engineering diagram of the system. Of course, the human must posses a certain amount of engineering knowledge. The analogous inputs in the computer understanding process are the computer representation of the system and the computer knowledge base. The human uses engineering knowledge and rationalization to produce a subsystem level model and the functional relations which interconnect the various subsystems. In the computer, the knowledge base and automated rationalization should also produce a subsystem level model with interconnecting functional relationships. The human uses the subsystem model to produce a behavior model which can be used for fault diagnosis or for design purposes. The computer, on the other hand, will perform similar tasks directly on the subsystem model (with appropriate functional relationships). The major goal of this part of the project at this point is the study of this process.

The most difficult part for the computer in this understanding procedure is discovering and representing the necessary functional relations between the parts of the subsystem model. Therefore, an important consideration is exactly what knowledge is needed in the computer knowledge base. The computer knowledge base should at least contain a definition of each component, including:

(1) component function,
(2) relationship with other components,
(3) device function,
(4) failure modes of the components,
(5) relative probabilities of various failure modes,
(6) possible effects of each failure mode on the rest of the system.

Further research will be done to determine the best way to discover and represent the functional relations between the parts of the subsystem model. More work is being done on the computer subsystem level model to allow it to include the functional relations between its parts. Solving these two problems will determine to a large extent how to apply the knowledge base and rationalization process to the computer model of the system. Having accomplished this, it will remain to write a CSA generator that uses the computer's model to produce a CSA diagram. An issue that merits future study is the ability of the system to add information to the computer knowledge base as it encounters new, undefined components.

4.2. Design Models in the Aircraft Engine Domain

The construction of models in the aircraft engine domain is of interest since such models will be of use in both the fault diagnosis procedure as well as the design process. What follows is a discussion of the use of a hierarchical model organization in the design process.

As an example, consider a refrigeration system. The refrigeration system can be separated into three separate areas:

(1) air flow removes heat from a coil (air flow is changed into refrigerant flow because of heat transfer through the coil).
(2) the refrigerant flows to another coil.
(3) heat is transferred to the refrigerant through the coil from an outside air flow (removing heat from the air flow).

This refrigeration problem reduces to a refrigerant flow problem and two air flow problems. The trick is to decide where the design problem is most conveniently divided.

The following six areas are a desirable classificatory partition for the simplification of the design problem:

(1) heat energy,
(2) electrical energy,
(3) magnetic motion,
(4) magnetic torque,
(5) mechanical motion (flow), and
(6) mechanical torque.

Let the abbreviation of each identifier signify the corresponding design area: for example, "mmot" indicates mechanical motion. Append an additional letter to provide a more specific formalization; for example, "mmotr" would indicate refrigerant flow, a type of mmot.

This partition can be used to specify the function of each part or component of a system. Provide an "input area" and an "output area" for each component in order to make identification of conversion components much simpler. For example, a conversion component could be the rotor of a motor, which converts magnetic torque to mechanical torque. Therefore, this particular component's input area would contain "magnetic torque" while its output area would contain "mechanical torque." Such input/output classifications are of help where a previously designed part is suitable, since it is not necessary to consider the internal workings of each subcomponent.

Although a compressor has many subsystems, one could consider it to have an electrical energy input and a flow output. A refrigeration expert can deal with this simplified (input/output) representation of a compressor without having to consider the compressor as a whole, thus not considering piston or motor design. This design area approach simplifies the design of a large system by dividing the problem into more tractable subproblems.

While it may seem obvious that a component may in fact be broken down into several subcomponents, this breakdown presents a representation problem. One method of dealing with this representation problem is the frame approach. A compressor frame would consist of an identifier (such as CMP1), input and output areas, and associated subcomponents. If there is more than one compressor, it is necessary to keep an index list of all components of this type, such as (COMP (CMP1 CMP2 CMP3)). Each of these would in turn have a unique description frame. What follows is an example of the top level of a compressor frame.

```
(cmp1 (eleng    25)
      (mmotr    30)
      (subparts (rotor1 coil3 piston7 drive5)))
```

This states that cmp1 takes in quantity 25 of electrical energy and produces quantity 30 of refrigerant flow. The subcomponents are rotor1, coil3, piston7, and drive5. The existence of cmp1 would be signaled on the index list:

```
(comp22 (cmp1 cmp2 cmp3 cmp4)).
```

The identifier comp22 identifies this particular type of compressor.

To begin the design process, the desired system must be specified. This specification takes the form of a list of subsystems.

```
(refsys (fanflow1 nil) (refflow 30) (fanflow2 nil))
```

The system identifier is refsys (refrigeration system). The three subsystems are fanflow1 (the coil that must release heat from the refrigerant), refflow (the refrigeration flow system) which must flow with value 30, and fanflow2 (the coil that removes heat from the refrigeration box). In order to begin designing a subsystem, it must have some associated value. Since the only subsystem with an associated value is refflow, the design process would begin with this subsystem.

```
(refflow (compr concoil exvalv evacoil pipeh pipel
          pipeq pipex) (mmotr))
```

This is essentially an available parts list. The design procedure would begin to look through this parts list until a component is found that can handle mmotr flow of 30, as specified in the design. Assuming that concoil is selected as the proper element for this design, control now returns to the design of fanflow1.

```
    (fanflow1 (concoil fans) (mmota))
```

Once concoil has been selected, the input airflow for that coil can be used as the air flow the fan must put out, and the fan can be selected based on its output area rating.

The original design can be completed by using the same chaining method in order to specify fanflow2. It is clear that in this example only one area, refflow, should have its value specified. If another area had also been specified, there is the possibility that no consistent design would have been possible within the constraints of the parts list.

4.3. Redundancy in Diagnosis

In some domains of interest diagnosis involves dealing with systems that are highly redundant. This part of the project is intended to investigate the ramifications of redundancy with respect to the diagnosis process.

4.3.1. The Ramifications of Redundancy on the Diagnosis Process

The question of the effects of redundancy on the fault diagnosis process is a relevant one considering that an airplane is a highly redundant system. At first glance, it might seem that redundancy might in fact hinder fault diagnosis due to possible fault masking, as well as the increased complexity of the total system generally associated with redundancy. Fault masking occurs when it is impossible to detect a failure (or a set of failures) simply by observing the outputs of the system (or a limited set of sensors). Such cases appear in digital circuits, which might have a fault masking capability due to use of quaded logic, duplicated logic with voters, or (logically) non-minimal circuit implementations. Output behavior of the circuit might appear to be normal when in fact multiple faults may indeed be present.

Actually, the situation is quite different in the airplane domain. In this domain, virtually every redundant subsystem has a set of sensors monitoring its operation. In most cases a failure or a number of failures will not seriously hinder an airplane's ability to fly; however, there should be an indication of which subsystem or set of subsystems failed. In a redundant system where proper operation of subsystems is easily verifiable, redundancy aids diagnosis since a failure can be isolated with less effort when some subsytems are obviously operating properly than when a single subsystem failure propagates through many subsystems. In other words, redundancy simply helps to reduce a possible failure set by localizing the effects of a failure.

Another side effect of system redundancy is increased capability of multiple fault diagnosis. A common diagnostic heuristic for non-redundant systems is the use of subsytem interdependencies during diagnosis. For example, if an engine malfunctions and the fuel pressure was low it is common to assume that the fault lies somewhere in the fuel system, unless the voltage on the fuel pump is low also. Any other failure in the engine not related to the fuel system would be ignored at this time. Since an (equivalent) redundant system would keep the fuel flow normal, engine failure as well as a single pump failure would be immediately apparent.

If reliability is excluded as a reason for introducing redundancy then only duplication of subsytems is necessary to dramatically improve the diagnosability of the system. The duplicate system will still aid in multiple fault diagnosis, provided that the (multiple) faults are not in the same subsystem. Increasing the multiplicity of replication only insures that a greater number of faults in the same subsystem can be tolerated before reducing the diagnosis problem to the non-redundant case.

The previous statement is only true in systems where the operation of a subsytem is easily verifiable. In the domain of digital systems the malfunction of a subsystem may not be easily detectable. In this case at least triple redundancy is needed to determine the faulty module. Even so, the subsystem designated as the "culprit" may actually be sound, while one of the subsystems considered sound is actually at fault. The probability of such a mis-diagnosis is small, but not negligible.

A necessary condition for redundancy-aided diagnosis is that all of the (redundant) subsystems must be observable. If only the result of the complete system's operation is observable then a number of failures can be totally masked thus making diagnosis no easier (or perhaps even more difficult) than in the non-redundant case. A system with reliablilty and ease of diagnosis as design criteria should have all the necessary sensors included to insure subsystem observability.

4.3.2. Estimation of System Tolerance to Multiple Failures

Another problem that can be considered in the context of redundant systems is the system's tolerance to a set of subsystem failures. This question can be easily answered if the system is represented by a directed graph where each link represents dependency and each node represents a subsystem. Reliability at a node is defined as a minimum of the multiplicities of all the immediate subsystems supporting it. For example, a fuel pump having two sources of power (one from an engine driven generator and another from an on-board battery) and three fuel lines from the tanks, would have reliabilty of two. An extra node called the "system node" is inserted into the network. This system node is supported by subsystems whose operation is necessary for the system to remain operational. Any subsystem failures are propagated through the network and if the system node reliability goes to zero the system is considered inoperative. This method allows determination of which particular sets of subsystem failures will render the system inoperative, as well as which sets of subsystem failures are tolerated by the system (i.e., still permit the system to operate normally).

5. References

[Teitelman78] W. Teitelman, *Interlisp Reference Manual*, Xerox PARC, Palo Alto, 1978.

[Mark80] W. S. Mark and G. E. B. Jr., The RUSgrammar Parsing System, GMR-3243, General Motors Research Laboratories, Warren, Michigan, April, 1980.

[Schmolze82] J. G. Schmolze and R. J. Brachman, Proceedings of the 1981 KL-One Workshop, 4842, Bolt Beranek and Newman Inc., Cambridge, Massachusetts, June 1982.

[Maran83] L. R. Maran, D. T. Spoor and D. L. Waltz, Encoding the Natural Language Meaning of Time, *Submitted to the Inter-disciplinary Workshop on Motion: Representaion and perception*, Toronto, Ontario, Canada, April 1983.

[McDermott82] D. McDermott, A Temporal Logic for Reasoning About Processes and Plans, *Cognitive Science 6*, 2 (1982), 101-155.

[Allen81] J. F. Allen, *Maintaining Knowledge about Temporal Intervals*, Dept. Computer Science, Univ. of Rochester, Rochester, NY, September 1981.

[Vilain82] M. Vilain, A System for Reasoning About Time, *Proceedings of the National Conferance on AI*, Pittsburgh, PA, August, 1982, 197-201.

[Waltz82a] D. L. Waltz, Event Shape Diagrams, *Proc. National Conf. on Artificial Intelligence*, Pittsburgh, PA, August 1982, 84-87.

[Waltz82b] D. L. Waltz, R. T. Chien and G. DeJong, An Expert Distributed Robotics System with Comprehension and Learning Abilities in the Aircraft Flight Domain, T-116, Coordinated Science Lab, Univ. of Illinois, Urbana, IL, July 1982.

[Group80] G. A. E. Group, *Aircraft Gas Turbine Guide*, General Electric, Cincinnati, OH, October 1980.

[Doyle78] J. Doyle, Truth Maintenance Systems for Problem Solving, Tech. Rept. Tech. Rep.-419, MIT AI Lab, Cambridge, MA, 1978.

[Stallman79] R. M. Stallman and G. J. Sussman, Problem Solving about Electrical Circuits, in *Artificial Intelligence: An MIT Perspective*, P. H. Winston and R. H. Brown (ed.), MIT Press, Cambridge, MA, 1979, 33-91.

[Forbus80] K. D. Forbus, A Study of Qualitative and Geometric Knowledge in Reasoning about Motion, M.S. Thesis, MIT AI Lab, Cambridge, MA, February 1980.

[Rieger75] C. Rieger, The Commonsense Algorithm as a Basis for Computer Models of Human Memory, Inference, Belief and

Contextual Language Comprehension, in Theoretical Issues in Natural Language Processing, R. Schank and B. Nash-Webber (ed.), ACL, Arlington, VA, 1975, 180-195.

[DeJong82a] G. DeJong, Automatic Schema Acquisition in a Natural Language Environment, Proc. National Conf. on Artificial Intelligence, Pittsburgh, PA, August 1982, 410-413.

[DeJong82b] G. F. DeJong and D. L. Waltz, Understanding Novel Language, WP-36, Coordinated Science Lab, Univ. of Illinois, Urbana, IL, July 1982.

[DeJong79] G. DeJong, Skimming Stories in Real Time: An Experiment in Integrated Understanding, Research Rept. 158, Yale Computer Science Department, New Haven, CT, 1979.

[Bondy76] J. A. Bondy and U. S. R. Murty, Graph Theory with Applications, North Holland, New York, 1976.

[Minsky75] M. L. Minsky, A Framework for Representing Knowledge, in The Psychology of Computer Vision, P. Winston (ed.), McGraw-Hill, New York, NY, 1975, 211-277.

6. Publications

DeJong, G., "An Overview of the FRUMP System," in Strategies for Natural Language Processing, W. Lehnert and M. Ringle (Eds.), Lawrence Erlbaum Associates, Hillsdale, NJ (1982).

DeJong, G., "Automatic Schema Acquisition in a Natural Language Environment," Proceedings of the 1982 National Conference on Artificial Intelligence, Pittsburgh, PA (1982).

DeJong, G. and Waltz, D., Understanding Novel Language, The International Journal of Computers and Mathematics (to appear 1983).

Maran, L. R., Spoor, D. T., and Waltz, D. L., "Encoding the Natural Language Meaning of Time," Submitted to the Interdisciplinary Workshop on Motion: Representation and Perception, Toronto, Canada, April 1983.

Pollack, J. and Waltz, D. L. 1982. "Natural language processing using spreading activation and lateral inhibition." Proceedings of the Conference of the Cognitive Science Society, Ann Arbor, MI, August 1982, 50-3.

Waltz, D. L. 1982. "Artificial Intelligence." Scientific American, 247, 4, October 1982, 118-33.

Waltz, D. L. 1982. "Event Shape Diagrams." Proceedings of the National Conference on Artificial Intelligence, Pittsburgh, PA, August 1982, 84-7.

Waltz, D. L. 1982. "The state-of-the-art in natural language understanding." In M. Ringle and W. Lehnert (eds.), Strategies for Natural Language Processing. Hillsdale, NJ: Erlbaum Associates, 3-36.

7. Personnel

There are three investigators on this project: Professor David Waltz (principle investigator), Professor Gerald DeJong and Professor R.T. Chien. Eight graduate students are contributing to the project: David Spoor and Raman Rajagopalan (advisees of Professor Waltz), Alberto Segre, Christian Debrunner and Paul Harrington (advisees of Professor DeJong), and Adam Pajerski, William Frederick and Michael Houghton (advisees of Professor Chien).

8. Interactions

Visitors

Michael Lebonitz
Columbia University (2/82)

Vincent Lum
IBM Research
San Jose, CA (3/82)

Jan-Olof Eklundh
Royal Institute of Technology
Stockholm, Sweden (4/82)

Charles Davidson
Chalmers University of Technology
Gothenburg, Sweden (7/82)

Allan Collins
Bolt, Beranek and Newman, Inc.
Cambridge MA (9/82)

C.R.D. Tatham and W.B.S. Owsianka
Central Computer and Telecommunications Agency
H.M. Treasury
London, England (9/82)

Michael Moran and John Vittal
Xerox Electro-Optical Division
Los Angeles, CA (10/82)

Arnold Beckman
Beckman Instruments
Pasadena, CA (10/82)

Jake Scherer
Rome Air Development Laboratory
Rome, NY (12/82)

Visits by D.L. Waltz

Marvin Denicoff
Office of Naval Research
Arlington, VA (1/82)

DARPA and ONR contractors meeting
Defense Advance of Research Projects Agency
Arlington, VA (1/82)

Program Committee Meeting (chairman)
National Conference on Artificial Intelligence (AAAI-82)
Massachusetts Institute of Technology
Cambridge, MA (4/82)

Brandeis University
Waltham, MA (4/82)

Bolt, Beranek and Newman, Inc.
Cambridge, MA (4/82)

National Conference on Artificial Intelligence (AAAI-82)
Pittsburgh, PA (8/82)

Computer Science Advisory Committee
National Science Foundation (9/82)

Longwood College
Farmville, VA (9/82)

Symantec Inc.
Sunnyvale, CA (11/82)

Stanford University
Palo Alto, CA (11/82)

Fairchild Advanced Research and Development Laboratory
Palo Alto, CA (11/82)

IBM Thomas J. Watson Research Laboratory
Yorktown Heights, NY (12/82)

IBM SRI
New York, NY (12/82)

Visits by G. DeJong

Research Initiation Review Panel
National Science Foundation
Washington, DC (6/82)

Cognitive Science Conference
Ann Arbor, MI (8/82)

National Conference on Artificial Intelligence (AAAI-82)
Pittsburgh, PA (8/82)

Coordinated Experimental Research Review Panel
National Science Foundation
Chicago, IL (11/82)

<u>Other Visits</u>

R.T. Chien, C. Debrunner, W. Frederick, M. Houghton, A. Pajerski, A. Segre
National Conference on Artificial Intelligence (AAAI-82)
Pittsburgh, PA (8/82)

C. Debrunner, P. Harrington
IEEE COMSAC-82
Robotics Tutorial
Chicago, IL (11/82)

9. Inventions and Patent Disclosures

There are no inventions or patent disclosures to report.