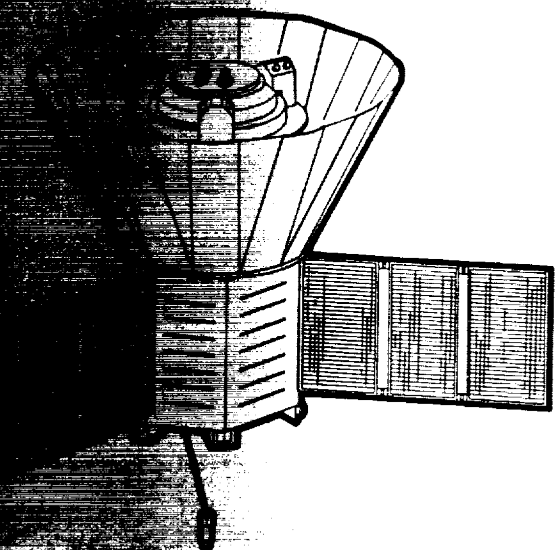
[illegible] Conference Publication 3068

[illegible] 1990 Goddard Conference on [illegible] Applications of Artificial Intelligence



Proceedings of a workshop series held at
NASA Goddard Space Flight Center
Greenbelt, Maryland
May 1–2, 1990

NASA

NASA Conference Publication 3068

1990 Goddard Conference on Space Applications of Artificial Intelligence

James L. Rash, *Editor*
Goddard Space Flight Center
Greenbelt, Maryland

Proceedings of a workshop series held at
NASA Goddard Space Flight Center
Greenbelt, Maryland
May 1–2, 1990

NASA

National Aeronautics and
Space Administration

Office of Management

Scientific and Technical
Information Division

1990

1990 Goddard Conference on Space Applications of Artificial Intelligence

Conference Committee

Peter Hughes, GSFC *(chair)*
Beth Antonopulos, GSFC
David Beyer, BFEC
Carolyn Dent, GSFC
Nancy Goodman, GSFC
Carl Hostetter, GSFC
Ed Luczak, CSC
James Rash, GSFC
Steve Weaver, CSC

PRECEDING PAGE BLANK NOT FILMED

Foreword

The fifth annual Goddard Conference on Space Applications of Artificial Intelligence is sponsored by the Mission Operations and Data Systems Directorate in cooperation with the American Institute of Aeronautics and Astronautics (AIAA), National Capital Section. The mission of this conference is to provide an opportunity for researchers and practitioners of artificial intelligence in the aerospace industry to share the results and experiences of their work. Much can be gained from such interaction. We anticipate that this conference will provide an effective forum for the exchange of ideas, interests, and results.

During the past decade, we have witnessed the maturation of artificial intelligence as it emerged from the isolation of research labs into the mainstream of the aerospace industry. However, although AI applications are becoming increasingly more common, many groups are still struggling with the problem of integrating artificial intelligence into operational environments. Addressing this problem has expanded our understanding of the intricacies of successfully using this rapidly advancing technology.

The technical papers contained in these proceedings are representative of the diversity of applications of artificial intelligence within the aerospace industry. They range from applied research projects to embedded operational systems and represent work from universities, private industry and various NASA centers. These papers give evidence -- and the participants in this area recognize -- that there is no more exciting or more potent combination of technologies than that which results from the application of artificial intelligence to support space exploration.

This conference would not have been possible without contributions from many people. First, I thank the authors, whose technical contributions were presented at the conference and recorded in this document. Second, I thank the invited speakers and panel session participants for taking the time to prepare their talks and engage in a realtime dialog with the conference attendees. Third, I thank the Abstract Evaluation Committee and the Technical Paper Evaluation Committee for their expert and diligent evaluations. Finally, I especially thank the members of the Conference Planning Committee whose support and dedication not only made this conference possible but also made the task of organizing it a rewarding and enjoyable experience.

Peter M Hughes

Peter M. Hughes
Chair
1990 Goddard Conference on Space Applications of Artificial Intelligence

Table of Contents

Planning and Scheduling

Diagnosis/Monitoring

Robotics/Intelligent Control

Special Topics

Planning and Scheduling

N90-22295

The Application of Connectionism to Query Planning/Scheduling in Intelligent User Interfaces

*Nicholas Short, Jr.**

Lokendra Shastri *

Abstract- In the mid nineties, The Earth Observing System will generate an estimated 10 Terabytes of data per day. This enormous amount of data will require the use of sophisticated technologies from real-time distributed Artificial Intelligence (AI) and data management. Without regard to the overall problems in distributed AI, this paper focuses on developing efficient models for doing query planning/scheduling in intelligent user interfaces that reside in a network environment. Before intelligent query/planning can be done, a model for real-time AI planning/scheduling must be developed. As Connectionist models (CM) have shown promise in increasing run-times, this paper proposes a connectionist approach to AI planning/scheduling. The solution involves merging a CM rule-based system to a general spreading activation model for the generation and selection of plans. The system was implemented in the Rochester Connectionist Simulator and runs on a Sun 3/260.

INTRODUCTION

The major mission of the National Space Science and Data Center (NSSDC) has been to archive and provide access to a wide variety of data from NASA's scientific experiments in the Earth and space disciplines. Historically, the NSSDC has been a centralized organization where all of the 2,500 online accesses per year for the over 125,000 tapes are sent to the Goddard facility (Green, 1989). While the volume of data requests can be currently handled, the arrival of many of NASA's new projects promises to cause not only a glut of data but also a large increase in number of data requests. This enormous amount will create a bottleneck at the centralized facility.

While an increase in the NSSDC's resources will assuage some of these problems, the authors believe that the size and complexity of the upcoming projects will require a distributed systems approach. For example, the NSSDC's data holdings will double every two years, reaching about 30 Terabytes by 1995 (Green, 1989). Moreover, projects like the Earth Observing System (EOS) will generate an estimated 10 Terabytes of data per day (Campbell, 1988). Considering that the NSSDC's current archive contains around 6 Terabytes (Green, 1989), it is doubtful that the current or future facility's resources can support the volume of requests

* Nicholas Short, Jr. is with NASA/Goddard Space Flight Center, Code 634, National Science Data Center, Greenbelt, Maryland 20771. He is also a graduate student at the University of Pennsylvania, Philadelphia PA 19104. Lokendra Shastri is a faculty member at the University of Pennsylvania.

without an introduction of new real-time distributed data management into the organizational structure.

Foreseeing this need, the NSSDC established the Intelligent Data Management (IDM) project, which has, as one of its goals, to develop advanced workstation tools for operation in a network environment. One component of this involves the use of intelligent user interfaces in a distributed architecture (Short, 1988). These interfaces will contain knowledge about the available resources and the access procedures not only from the NSSDC but from other archives as well.

One way to reduce the number of requests would be to guarantee that the queries to the facility are both accurate and supported by existing data. That is, if tools can be provided to the user that will allow him to develop intelligent queries at his own facility, then fewer ill-formed queries will be sent to the data center.

As is well known to researchers in real-time systems and database management, a critical ability required will be the need to plan and coordinate the various data access and manipulation tasks that are required to support a particular user's data goal. For example, suppose a user requires data sets A,B,C, and D and archives 1,2, and 3 contain some subset of the needed sets. Specifically, suppose

1) archive 1 contains A and B;

2) archive 2 contains A, B, and C;

3) archive 3 contains C and D.

Now, the obvious access plan would be to query either archive 2 and archive 3 or archive 1 and archive 3. The choice between these options will be determined from constraints such as network loads, the request arrival rates to the archives, processor loads at a particular archive, etc. All these choices must be resolved in real-time in spite of large number of expected data requests.

This problem of solving tasks based on constraints has been addressed by the planning sub-field of Artificial Intelligence, as well as the real-time systems field (Stakovic, 88). It would be apparent that solving the real-time constraints in distributed query planning will require a solution to the planning/scheduling problem.

Unfortunately, intelligent real-time planning in the AI field is only now beginning to be addressed by researchers. In this paper, we will explore the limitations of classical AI planning, propose a solution, and discuss some of the applications to the data management field. We hope that a minimization of requests to an archive can occur through the intelligent formulation and execution of queries through planning/scheduling.

QUERY PLANNING/SCHEDULING AND REAL-TIME SYSTEMS

Not surprisingly, scheduling of tasks for real-time systems cannot be based on algorithms which just try to satisfy deadlines. Stankovic states that

"The goal is to find optimal static schedules that minimize the response time for a given task set....The system is often highly dynamic, requiring on-line, adaptive scheduling algorithms" (Stankovic, 1988).

In fact, as these algorithms are NP-hard, the solution must be heuristic, forcing the problem into the AI domain (Zhao, 1987).

Any hope of solving this scheduling problem along with planning may have been thwarted by Chapman's proof that efficient, general-purpose planning is undecidable (Chapman, 1987). Yet, humans can solve the problem by, as Chapman states, "...improvisation, doing something easy and debugging the result when it fails " (Chapman, 1987).

This suggests that a solution may be to store numerous fixed plans, determine which plans are applicable to a given situation, and execute these plans efficiently until an error in execution occurs. Many trials (i.e., plan attempts) will be executed before the correct final plan is determined. That is, incremental planning must be done.

The incessant need to plan/replan will cause havoc for the scheduling algorithms which are trying to minimize response time for the systems. This delicate balance between response times and planning can be maintained by utilizing the most efficient method for heuristic planning/scheduling.

Connectionist Models (CM) have shown time reductions in several classical Artificial Intelligence (AI) algorithms (Shastri, 1989), but few CMs have been applied to the planning field (Whithead, 1989; Blelloch, 1986). In this paper, we present an initial attempt to move some of the functions of a classical planner into a CM. The idea involves merging spreading activation over a semantic task-net and rule-based inference into a CM to aid in the selection and generation of plans (Hendler, 1988; Shastri, 1989).

From the network system's perspective, The CM planner will reside on each node and will exist in the node's intelligent user interface. The user interface's knowledge-base will contain information about interactive problem solving among other network nodes. While this distributed AI problem is beyond this paper's scope, we will focus on the model of a node's CM planner without regard to specific details about cooperative planning over the network.

The role of the CM in a particular node's data management system will be to interface the high-level, symbolic components to the standard database models, as described in (Short, 1988). More specifically, expert database advisors, natural language front-ends, and graphics interfaces will comprise the high-level while database management systems will form the low-level.

PLANNING DESCRIPTION

In general, planning can be broken into two distinct phases: plan generation and planning decisions (Charniak, 1985). Plan generation resembles a deductive problem in that, given a task (goal) and a situation (database), we achieve that goal by solving any number of subtasks (subgoals). The product of plan generation is an "and/or" graph where the "and" branches correspond to ordered steps in a plan and the "or" branches match alternative plans.

Planning decisions, on the other hand, consist of two phases: plan coordination and plan selection. In the former, the partial-order produced by the plan generator is converted to a total-order. This is done by detecting plan failures that occur with some orderings of the "and/or" graph. In the latter, plan selection involves searching through a plan library to find the best alternatives (i.e., for the "or" branches). In this phase, the proper selection of plans can aid in the reduction of the "and/or" graph and, hence, the run-time of the planner.

In fact, if it can be shown that, while the plan generator is executing, a particular choice of plan can never be used given the current choice of plans, then no further reduction of that plan is required. For example, suppose that our planner is flown out to California on a trip and that while in California, it decides to buy a gun. If the planner has some prescience about airport security, it will realize that a return flight is impossible, because carrying a gun through a metal detector is illegal. There will be no need to reduce the subgoal "fly home" and an alternative plan like "drive home" could then be selected for reduction.

Analogously, some choices of plans could be chosen over others based on the current situation. For instance, suppose our planner wanted to determine the amount of deciduous forestation in Maryland and it contains an entry in a database for such information. In this situation, it would be better for the planner to select the alternative "database-query" as opposed to "calculate-from-raw-image-data." Choosing the former would save the planner from having to find the Landsat data, find an efficient processing environment (e.g., the MPP), determine the correct algorithms, transfer the data to the MPP, etc.

The output from the planner is a the total-order of tasks to be executed in order to achieve the goal. Each step in the total-order is a primitive task that corresponds to a leaf in the "and/or" graph and is defined a priori. One phase not included in the above discussion is the use of execution monitoring to provide feedback to the planner about its choice of plans. This module will notify the planner when error recovery routines must be invoked. Moreover, it can provide statistical information about outcomes in order to aid the plan selector.

RULE-BASED CONNECTIONISM

One of the first attempts to reduce the the run-times of planners was Hendler's Scraps model (Hendler, 1988). Scraps kept the plan generation and coordination modules in a symbolic logic-based system (i.e., NASL), while the plan selector was moved to a CM-like method using spreading activation over a task-network.

Although spreading activation was introduced by psychologists (i.e., by M. Quillian) as a cognitive model, it has proven useful to AI in reducing the costly time of unification of the rules to semantic networks. In general, the spreading activation is a bi-directional, breadth-first search over a semantic net. Beginning at two nodes, two searches are conducted by having each search mark its neighboring nodes as visited until the two searches intersect via a path. Once a path is found, it is returned to the logic system for unification against the rule-base. If unification fails, then more paths can be examined, as the spreading activation algorithm will continue independently of the logic system. Spreading activation in these terms can be viewed as a fast-subsetting mechanism of the semantic net. Of particular note, the nature of spreading activation makes it very amenable to implementation on parallel machines that act in background to the logic system.

In Hendler's model, a variant of spreading activation, called marker passing, is used where instead of just marking nodes as visited, nodes receive and save complex messages as marks. When coupled with the plan generator, the marker passer looks through memory to determine which plans the plan generator should reduce. That is, the marker passer would return paths to a plan evaluator which would either rule in or rule out choices. Specifically, the plan evaluator consisted of a set of heuristics which would reject or accept paths a viable before notifying the plan generator about plan choices.

The problem with Scraps, as with all marker passing systems, is that too many irrelevant paths can be returned. For example, in Charniak's WIMP system, only two paths out of 40 returned paths were usable from a semantic net of 75 nodes and 255 facts (Charniak, 1986). Hence, the path evaluator represented a bottle-neck that could make the time savings negligible.

One solution to the problem would be to move the plan generator into a CM in the hopes of getting rid of the plan evaluator. Unfortunately, attempts at moving more functionality into the planner have been limited due to what is commonly referred to as the "variable binding problem." For example, Hendler states

"Given a plan for 'MOVING X TO Y,' his (Blelloch, 1986) system must build special network components for each possible move that could be made. The range of X and Y must be predefined and the system can only plan on those. For example, given N blocks we generate the 2(1 + 2 + ...N)(N -1) plans for MOVE-A-TO-B, MOVE-A-TO-C, etc. These are then the only operators usable, and new blocks cannot be added without changing the system" (Hendler, 1988).

Another solution comes from the recognition that plan generation resembles logical inference. That is, if we can modify a reasonable CM deductive-retriever, then both the plan generator and marker passer could execute simultaneously in the CM. In fact, this paper presents a limited step at realizing this goal.

Basically, the approach is to modify the rule-based inference of (Shastri, 1989) to work with a spreading activation algorithm. In general, this is done by organizing the plans and goals into rules and the constants into a semantic net. Solutions to goals are then done by simultaneously proving a goal using the Shastri & Ajjanagadde model and spreading the activation over the constant semantic net. When a path is found that could rule-out a subgoal or plan, an inhibitory activation is sent over to the rule-based side of the network. That is, if a goal or task is ruled-out and the rule-based portion of the net has not tried proving it, then the inhibition will prevent the rule-based side from working on that subgoal.

Unfortunately, there can occur situations in which the rule-based side has already started proving the subgoal that was ruled-out. In this case, the inhibitory activation wili send notification of its inhibition to the parent goals recursively up the "and/or" graph. Thus, both the rule-based model and the marker passer are in a race-condition. From the point of view of the planner, this presents no problem. However, the goal of reducing some of the time spent in plan generation may not have been met.

THE RULE-BASED CONNECTIONIST MODEL

In this connectionist model, there exists several types of units which correspond to predicates, arguments, etc. For each unit, several types of sites cluster the input

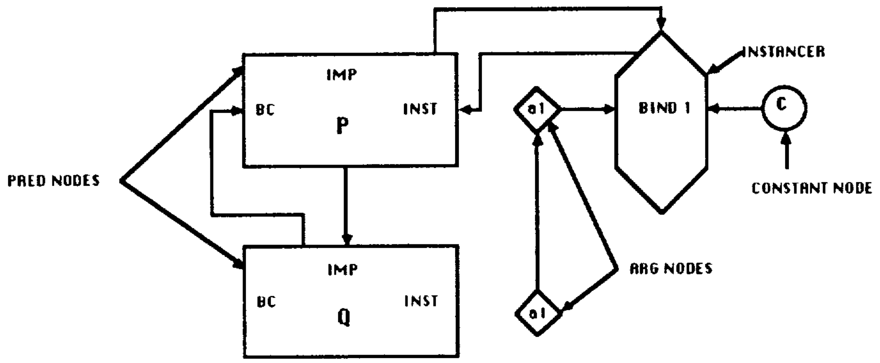


FIGURE 1 CONNECTIONIST ENCODING OF "FOR EVERY X, P(X) IMPLIES Q(X)"

connections from other units and modulate the inputs depending on the site characteristics. Connections between units are made to simulate both the database of assertions and the implications of the rules. Below, we provide an abstract description of the representation and reasoning in the rule-based system. For a complete discussion on the expressiveness of the rule-based system, see (Shastri, 1989; Shastri 1990).

In fig. 1, predicates are represented as rectangles, the associated arguments are represented as diamonds, and constants are represented as circles. For every predicate, there exists a number of associated argument nodes for every variable slot. The hexagons, called instancers, represent instantiated predicates. Suppose we wish to represent the rule:

(Every (x) (P x) => (Q x)) (*)

Also, suppose that we also know (P c) for some constant, c. If we wished to determine whether (Q c) were true, then we would activate the node representing Q, the argument node corresponding to Q, and the constant node c, allowing the network to run until (Q c) is proven true or false.

Now, each of these nodes is based on a node type called a binary threshold unit (BTU). That is,a BTU will output a 1 if any of its input values equals one. Otherwise, it will output 0. The key to controlling the spreading of activation is to make the activation of these BTUs phase sensitive.

The phase interval structure is defined by the query structure. In other words, the spreading activation is controlled by a clock where each cycle is broken into a fixed number of phases. The number of phases is dependent upon the number of bound arguments a the query. For example, one phase would be required to prove (R c) or (Love c y), for some constant, c, and variable, y.

A node type is partially characterized by the phase for which it can be activated. So, a constant node is active in the phase for which it was initially activated. For example, if the query (loves John Mary) were posed, the John and Mary constants would always be active in the first and second phase, respectively. Similarly, an argument node becomes active in a phase i if it receives input from phase i in the previous cycle.

Predicate and instancer nodes are more complicated and are abstractions of BTUs. First, a predicate (pred) node contains three sites, IMP, INST, and BC, which collect all the connections from other nodes. Instead of just two states as in the BTU, the pred node has three internal and output states. The internal states are Inert, Enabled, and Active and the corresponding output states are 0, low, and high. The pred node changes state from Inert to Enabled if its BC site receives low or high input and from Enabled to Active if its INST site receives at least a low input or its IMP site receives a high input.

Second, an instancer node contains an Enable site and n bind sites that correspond to the arg nodes in the instantiated predicate. The Enable site receives input from the instantiated predicate's output link, while each of the bind sites receive input from both the arg nodes of the instantiated predicate and the corresponding constant. The instancer node becomes active at the end of a cycle if every bind site receives input from both the corresponding argument node and constant node. If only either the argument node or the constant node sends activation to the same site, then the instancer cannot be activated. Once active, the instancer node sends its output to the INST site of its pred node.

Rules are encoded by making connections among the aforementioned node types. That is, implication is enforced by making links between the pred nodes and arg nodes of the antecedent and consequences of the rule. If a variable in the consequence occurs in the antecedent, then a link is made from the consequence's arg node to the antecedent's arg node. Also, there will exist a link from the consequence's pred output to the antecedent's BC site and a link from the antecedent's pred output to the IMP site of the consequence node. So, for the rule (P x y) => (Q x), Q will have one arg node which connects to the first arg node of P's two arg nodes.

Then, when all the rules of the knowledge base are compiled into this formalism, the corresponding network forms a type of directed acyclic graph (DAG). The leaves of this DAG correspond to those antecedents which correspond to either asserted facts or antecedents which require no proof. Answering a query then corresponds to sending activation from the query's pred node backwards on the DAG until the leaves are reached. Once the leaves become active, activation is sent back along the IMP links to the original query's pred node. If the query pred node receives activation back, represented via the state of the pred node, then the query is considered true, otherwise it is considered false. Notice that the complexity of the proof is then twice the length of the longest path in the DAG.

For an example, suppose we wanted to prove (Q c) from (*). Because there is just one bound constant,c, in the query, there would be just one phase per cycle. In phase 1 of cycle 1, the Q pred node's state would be set to Enable and the arg node and constant node c would be set to 1. At phase 1 of cycle 2, P's pred node will be enabled by the link from Q's output and P's arg node will be enabled by the link from Q's arg node. It will then send output to the Enable site of the instancer node corresponding to the fact (P c). At this point, since both P's arg node and the c node are sending output to the instancer node, the instancer node is activated and sends output to back to the P pred node. This causes the P node to go from a state of Enable to Active. When this occurs, high activation is sent to the IMP site of Q. This causes Q to go from a state of Enable to Active suggesting that the proof worked. Had we not had the fact that (P c) exists, the proof would have failed in the second cycle because the Bind site of the instancer node would not have allowed P to activate. That is, its Bind site would have been receiving activation from the arg node only.

SPREADING ACTIVATION SUBSECTION

THE SPREADING ACTIVATOR

Before getting into the details of the interaction between the spreading activator and the rule-based CM approach, the implementation of the spreading activator in the CM simulator will be described.

Generally, the implementation is a simplification of Hendler's marker passer. In Hendler's model complex markers consisting of fields like origin, fromnode, formula, zorch, etc. are passed from node to node. Because these markers violate the CM assumption that simple messages are passed, only one field, zorch, is used in the communication. Zorch is an attenuation mechanism that dampens the spreading of activation through the net. That is, as each mark is passed, the zorch factor is decreased by dividing it by the degree at each node. When zorch falls below a certain threshold, marking is stopped.

To start the spreading activation, the constants from the bound arguments in the initial goal are marked initially. In terms of the simulator, the unit's state is set to a value MARK and the potential and output are set to the initial zorch. To decrease zorch, the output links are weighted with the degree of the node. Zorch is then reduced at the site "MP" by dividing the link value by the weight (i.e., neighbor_num * 1000). If two zorch factors enter the site at the same time, then the smaller of the two is chosen.

In addition to reducing zorch, the site function at "MP" flips the pointer back to the originator of the activation. This is done so that the original path can be recovered when an intersection is found. Using this method avoids the problem of looping that Hendler's algorithm had to solve.

PATH EVALUATION

As aforementioned, Hendler's algorithm uses a set of heuristics to reduce the number of paths. When a path is returned from the marker passer, each heuristic examines the path to see if it is relevant to planning. Specifically, the five heuristics used are:

1) **QUICK REJECTION**: reject the paths that have already been examined,

2) **DEALING WITH DEMONS**: execute a demon when it occurs in the path,

3) **DEALING WITH PERCEPTUAL FLAGS**: "rule in" any tasks on a path that contains a perceptual flag,

4) **DEALING WITH FAIL FLAGS**: "rule out" any tasks on a path that contains a fail flag,

5) **DEALING WITH PLAN INTERACTIONS**: rule-in/rule-out plans that affect other choices in plan generation (e.g., the California plane trip example).

Of particular note, demons are rules that recognize certain conditions and interrupt the planner in order to add/modify plan steps. They are often used when a particular event must occur now, instead of later in the plan execution phase.

In addition to demons, flags are notes that are asserted into the semantic net when an important property has occurred. For example, when the planner is holding a gun, a flag is asserted into the net by the forward chaining rule

(-> (POSSESS ?x ?y) (FLAG ?y PERCEPTUAL (POSSES ?x ?y)

The assertion (FLAG 'gun PERCEPTUAL (POSSESS 'planner 'gun)) could be used by the path evaluator to rule-in the "shoot oneself" plan. Similarly, a fail flag like (FLAG FAIL (ON-STRIKE (CHEF ?x))) could rule-out a plan for a chef to cook a meal.

In our system, heuristic 1,3, and 4 can be replaced by the scheme discussed below. Although 2 and 5 may be possible in this scheme, they were not addressed.

First, heuristic 1 is implemented by a combination of the back links and spreading activation back from an intersection node. In detail, when two paths intersect, the intersecting node begins sending information back to the starting nodes. As this activation crosses the marked nodes, it checks the node-type, defined by the set membership in the simulator, to determine if activation should be sent to the rule-based portion. Since the rule-based nodes need only be excited or inhibited once to rule-in or rule-out a path, an activation crossing the marked nodes for a second time will have no effect on the corresponding rule-based node. Hence, duplicate paths are irrelevant.

The type-checking is actually implemented by a connecting link from the marked node to the corresponding node in the rule-based section. Details of how the rule-based section behaves will be discussed in the next section.

Second, heuristic 3 and 4 are implemented in a similar manner. Like Hendler's algorithm, flags are asserted into the net. However, when the marker passer crosses a flag, it checks the node type. If the node type is a fail flag, the zorch is negated and passed to its neighbors. This is done so that when an intersection occurs, the system knows that a rule-out should occur. That is, at an intersecting node, instead of sending a positive activation back to the origins, a negative or inhibitive value is returned. This tells the various nodes along the return path whether to excite or inhibit the corresponding rule-based nodes. Since this system only "rules in" or "rules out" plans, a positive activation implies a "rule in", whereas a negative implies a "rule-out."

RULE-BASED PLAN GENERATOR

Unfortunately, the CM rule-based implementation had to be modified to handle the interaction between it and the spreading activator. This amounted to changing many of the unit functions, site functions and behavior of the constant nodes.

First, in order for the variable bindings to work in the rule-based section, they must be activated in the phase corresponding to their position in the query. This will, however, disrupt the spreading activation as these constant nodes are participating in both plan generation and spreading activation over the constant task-net. That is, if a constant node activates in its phase it will not only send activation to an instancer node but also another neighboring constant. The neighboring constant will mistake that activation as zorch.

The solution to the problem is to notice that the binding of constants occurs only in the first cycle. So, if spreading activation is delayed until the second cycle, the first cycle can be used to notify the instancer node about the phase in which they should be activated. In other words, another type of instancer node is created. This node records the phase in which it receives activation in the first cycle. Acting independently from the constant nodes, it then activates in its respective phase for the rest of the cycles.

Second, in order to rule-in/rule-out plans, a link exists between the constants in the spreading activator and the rule-based section. That is, the constants consist of three types: constants, tasks, and flags. This semantic net is organized like a task-oriented hierarchy as in fig. 2. For each node of type "task" (henceforth called task constant) a link is made from it to the corresponding pred node in the rule-based section. So, for each plan/task/etc. two nodes are required instead of the one used in the original rule-based CM. So, when activation from a returned path crosses the task constant, the pred node receives activation from the task constant as to whether to activate or shut down.

IMPLEMENTATION DETAILS

The model was implemented in the Rochester Connectionist Simulator (RCS) on a Sun 3/260 workstation. The RCS was chosen for its portability to numerous machines including a parallel machine, implementations in both Suntools and Xwindows, and its graphics interface.

A lisp interface to the planner was written to allow access from various expert system shells, including the Automated Reasoning Tool and the Advice Taker/Inquirer (ATI) (Cromp, 1988). Eventually, the ATI will be used to enter in plans from an expert for execution in the connectionist planner. Both the ATI and the connectionist planner could reside on several nodes on the network, where each node will contain heuristic knowledge about network resources, network traffic, and access procedures.

EXAMPLE

Suppose that the intelligent user interface on machine A has determined that the user wishes to "get" a file which exists on another machine B. To illustrate how the spreading activator could stop plan generation, suppose that the other machine's "get" command has been disabled. Because of this, initiating the file transfer protocol would be useless and we would want the planner to avoid reducing the ftp command.

Specifically, the following plan library solves this problem:

```
(To-do (user-request ?dataset ?location)

        (DataAccessPlan ?dataset ?location))

(Plan (DataAccessPlan ?dataset ?location)

        (steps (check-net-node-status ?location on)
```

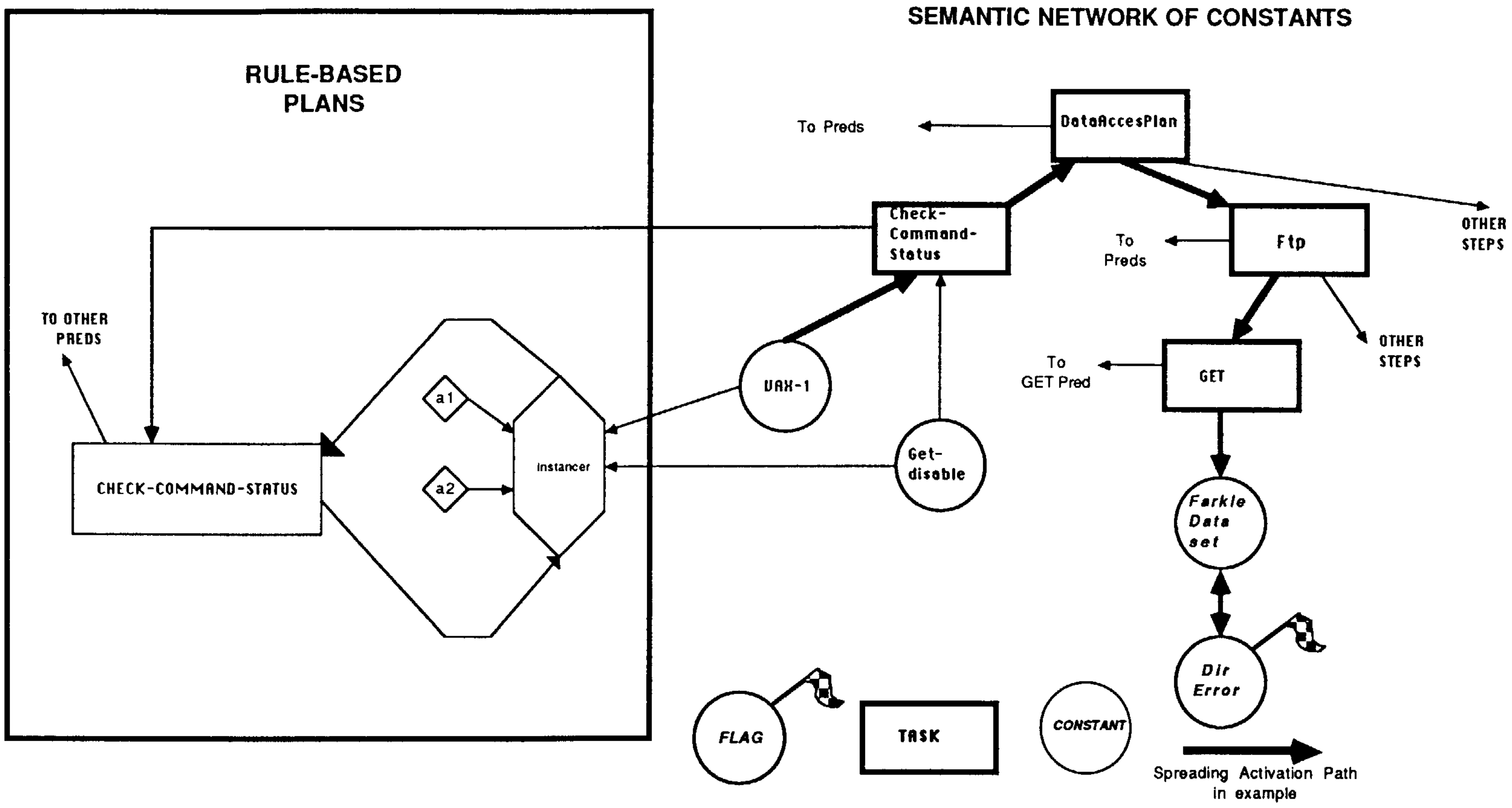


FIGURE 2. PLAN GENERATION AND SPREADING ACTIVATION ARENA

```
        (ftp ?dataset ?fromloc ?toloc)))

(Plan (ftp ?dataset ?floc ?toloc)

        (steps (open ?floc)

               (open ?toloc)

               (get ?dataset)

               (close ?floc)

               (close ?toloc)))
```

The connectionist implementation is shown in fig. 2 Because machine B's get is down, a flag is placed in the constant net. To solve this problem, the query

```
    (solve ‘(user-request ‘FarkleSet ‘Vax-1))
```

is posed to the connectionist net. Because there are two arguments, we have two phases for every clock cycle. After the first clock cycle, the spreading activation is started in the semantic net on the Vax-1 and FarkleDataSet nodes. Eventually, a path will be found between the DirError node and Vax-1, causing negative, inhibitory values to be sent from the intersecting node back to the starting nodes. When the inhibitory scalars cross the Get node, inhibition is sent to the Get pred node on the rule-based side. This will shut off the Get pred, causing negative scalars to be sent back to parent pred nodes via the implication links.

Of particular note, many of the details for the choice of constant node in spreading activation have been left out. See (Hendler, 1988) for a better description.

CONCLUSION AND FUTURE DIRECTIONS

The advantages of moving the plan generator into rule-based CM are threefold. First, we have shown how to avoid part of the path evaluator in order to stop the bottle-neck between plan generator and plan selector. Second, we have reduced the expense of plan generation by going to the linear run-time of the rule-based CM. Lastly, we can still use our hierarchical representations while ignoring the implementation question.

Unfortunately, not all of Hendler's functionality was achieved. For instance, Rule-in in our model is somewhat meaningless in the CM. Because there is no way of knowing which argument nodes in the middle of the "and/or" graph will be activated by lower argument nodes, the CM planner must wait for the activation to come up. Rule-in's only use is to help the plan coordinator in choosing an alternative. One possible addition could be to have the ruled-in pred node laterally inhibit its sister alternatives. Then, when activation reaches the ruled-in node, values will propagate only through that node. Hence, no nodes are unnecessarily activated.

Much work still needs to be done to make this a viable model for planning. For example, a plan coordinator must be integrated naturally, as (Whithead, 1989) has suggested. Finally, if not all of the planner can be moved into a connectionist model, then the simple planning could be reserved for the CM, while complex planning could be done in a classical symbolic planner. In other words, the planner first tries to plan and execute a plan sequence using the CM. If that fails, it then passes partial information up to the symbolic planner which uses that to generate a more complicated plan. This is, of course, exactly what Chapman suggested as a solution to the general planning problem.

While this research may not be significant in the "short-run" to NASA's data management problems, we argue that with NASA's appropriation of several parallel machines, connectionist models in general have the propensity to increase the efficiency for AI requirements in intelligent user interfaces. The major benefit of these models is that much of the classic AI algorithms (e.g., back-chaining) can be kept without any loss, as is not the case with more standard neural net approaches.

REFERENCES

Blelloch, G., (1986),"AFL-1: A Programming Language for Massively Concurrent Computers," MIT AI Laboratory, Technical Report AI-TR 918.

Charniak, E. (1986), and McDermott, D. ,*Introduction to Artificial Intelligence*, Addison-Wesley Publishers.

Charniak, E. (August, 1986), A Neat Theory of Marker Passing, *Proceedings from the Fifth National Conference on Artificial Intelligece* (pp. 584-88), Philadelphia, Pennsylvania.

Campbell, W., Short Jr., N. and Treinish, L.,(May, 1989) "Adding Intelligence to Scientific Data," *Computers In Physics..*

Chapman, D., (1987) "Planning for Conjunctive Goals," *Artificial Intelligence 32..*

Goddard, N., (1987) "The Rochester Connectionist Simulator: User Manual," Dept. of Computer Science, University of Rochester.

Cromp, R. (1988) "The Advice Taker/Inquirer, A System for High-Level Acquisition of Expert Knowledge, *Telematics and Informatics*, 5(3), pp. 297-312.

Green, J. (1989), "The New Space and Earth Science Informations Systems at NASA's Archive," Accepted for publication in *Government Information Quarterly.*

Hendler, J., (1988) *Integrating Marker-Passing and Problem Solving: A Spreading Activation Approach to Improved Choice in Planning*, Lawrence Erlbaum Associates, Publishers.

Shastri, L., (1988) *Semantic Networks: An Evidential Formalization and its Connectionist Realization,* Morgan Kaufmann Publishers, Inc.

Shastri, L. and Ajjanagadde V., (January1989) , "A Connectionist System for Rule Based Reasoning with Multi-Place Predicates and Variables," Computer and Information Science, MS-CIS-8905, Univ. of Pennsylvania.

Shastri, L. and Ajjanagadde V., (January1990) ,"From Simple Associations to Systematic Reasoning: A Connectionist Representation of Rules, Variables, and

Dynamic Bindings," Computer and Information Science, MS-CIS-90-05, Univ. of Pennsylvania.

Short, Jr., N. and Wattawa, S., (December 1988) "The Second Generation Intelligent User Interface for the Crustal Dynamics Data Information System," *Telematics and Informatics, 5(3)*, pp. 253-67.

Stankovic, J., (October 1988) "Misconceptions about Real-Time Computing," *IEEE Computer*, pgs. 10-19.

Whitehead, S. and Ballard, D., (1989) "Connectionist Designs on Planning," 1989 Summer Workshop on Connectionism, Carnegie-Mellon University.

Zhao, W., Ramamrithham, K., and Stankovic, J., (May 1987) "Scheduling Tasks with Resource Requirements in Hard Real-Time Systems," *IEEE Trans. Software Eng.*, Vol. SE-12, No. 5, pgs. 564-577.

N90-22296

RESOURCE ALLOCATION PLANNING HELPER (RALPH): LESSONS LEARNED

Ralph Durham, Norman B. Reilly, Joe B. Springer
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California
(818) 354-6316
FTS 792-6316
FAX (818) 393-4100

ABSTRACT

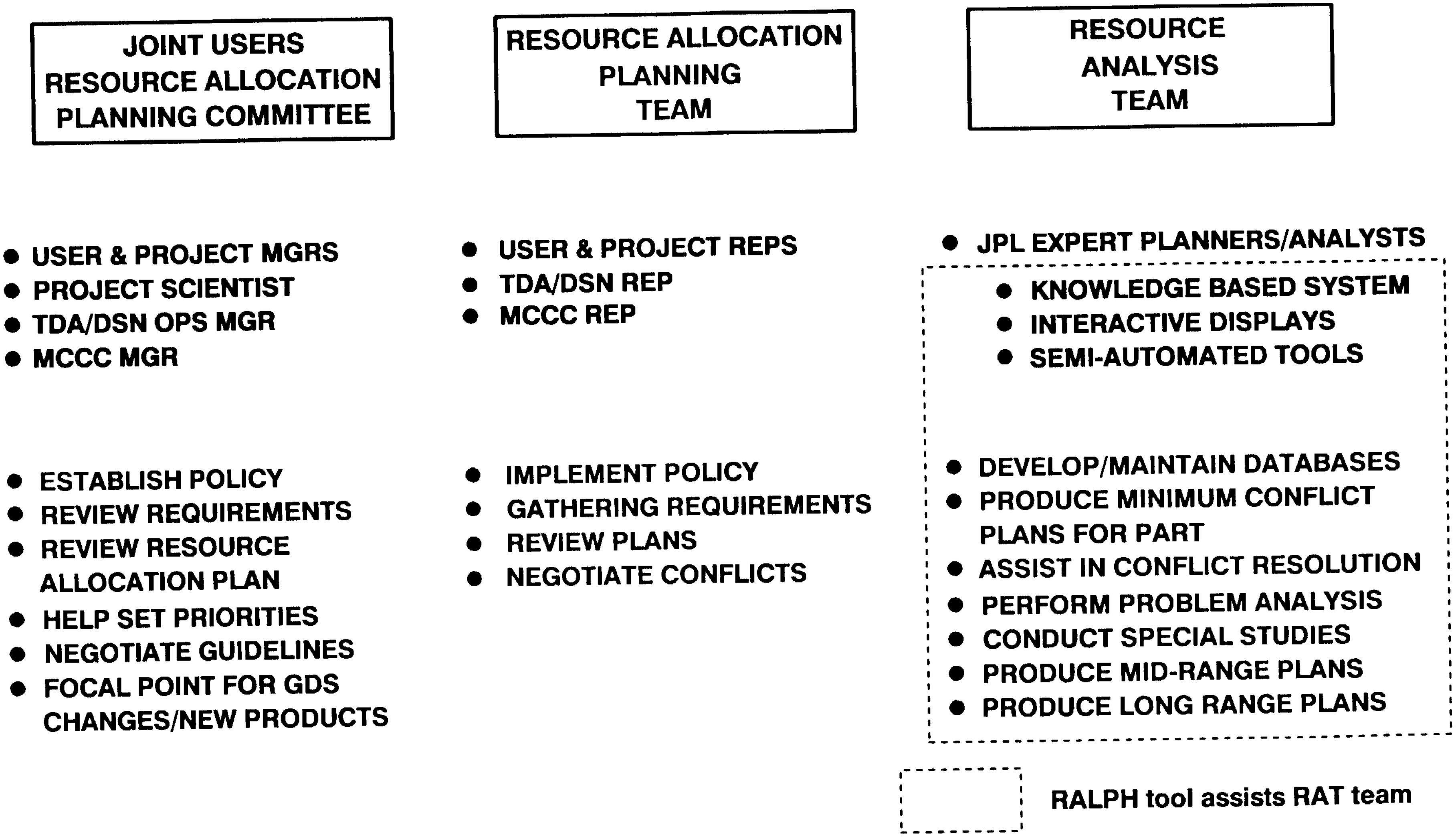
The Jet Propulsion Laboratory's (JPL) Resource Allocation Process incorporated the decision making software system RALPH into the planning process four years ago. The current principal task of the Resource Allocation Process includes the planning and apportionment of JPL's Ground Data System composed of the Deep Space Network and Mission Control and Computing Center facilities. The addition of the data-driven, rule-based planning system, RALPH, has expanded the planning horizon from eight weeks to ten years and has resulted in significant labor savings. Use of the system has also resulted in important improvements in science return through enhanced resource utilization. In addition, RALPH has been instrumental in supporting rapid turn around for an increased volume of special "what if" studies.

This paper briefly reviews the status of RALPH and focuses on important lessons learned from the creation of an highly functional design team, through an evolutionary design and implementation period in which we selected, prototyped and ultimately abandoned an 'AI' shell, and through the fundamental changes to the very process that spawned the tool kit. Principal topics include proper integration of software tools within the planning environment, transition from prototype to delivered software, changes in the planning methodology as a result of evolving software capabilities and creation of the ability to develop and process generic requirements to allow planning flexibility.

Also examined are strengthening of resource allocation techniques enabling implementation of effective conflict resolution strategies through an understanding of mission flexibility in the context of resource capacity/availability, characteristics and constraints, and techniques enabling early forecasting of resource loading to permit mission design changes. Finally, we present a discussion of a design which provides the ability to easily alter resource and requirements data tree structures to provide a problem-independent scheduling system applicable to a wide range of scheduling problems.

INTRODUCTION

During the last four years, the Resource Allocation Planning Helper (RALPH) has become an integral part of the Ground Data System Resource Allocation Process (RAP) at the Jet Propulsion Laboratory. (see Figure 1) As a result of the experience of users from the flight projects, the Resource Analysis Team and ground-based radio astronomy, both the process and the RALPH tool kit have begun to change and mature.[1] The experiences gained during this period of transition have provided some interesting



JOINT USERS RESOURCE ALLOCATION PLANNING COMMITTEE

- USER & PROJECT MGRS
- PROJECT SCIENTIST
- TDA/DSN OPS MGR
- MCCC MGR

- ESTABLISH POLICY
- REVIEW REQUIREMENTS
- REVIEW RESOURCE ALLOCATION PLAN
- HELP SET PRIORITIES
- NEGOTIATE GUIDELINES
- FOCAL POINT FOR GDS CHANGES/NEW PRODUCTS

RESOURCE ALLOCATION PLANNING TEAM

- USER & PROJECT REPS
- TDA/DSN REP
- MCCC REP

- IMPLEMENT POLICY
- GATHERING REQUIREMENTS
- REVIEW PLANS
- NEGOTIATE CONFLICTS

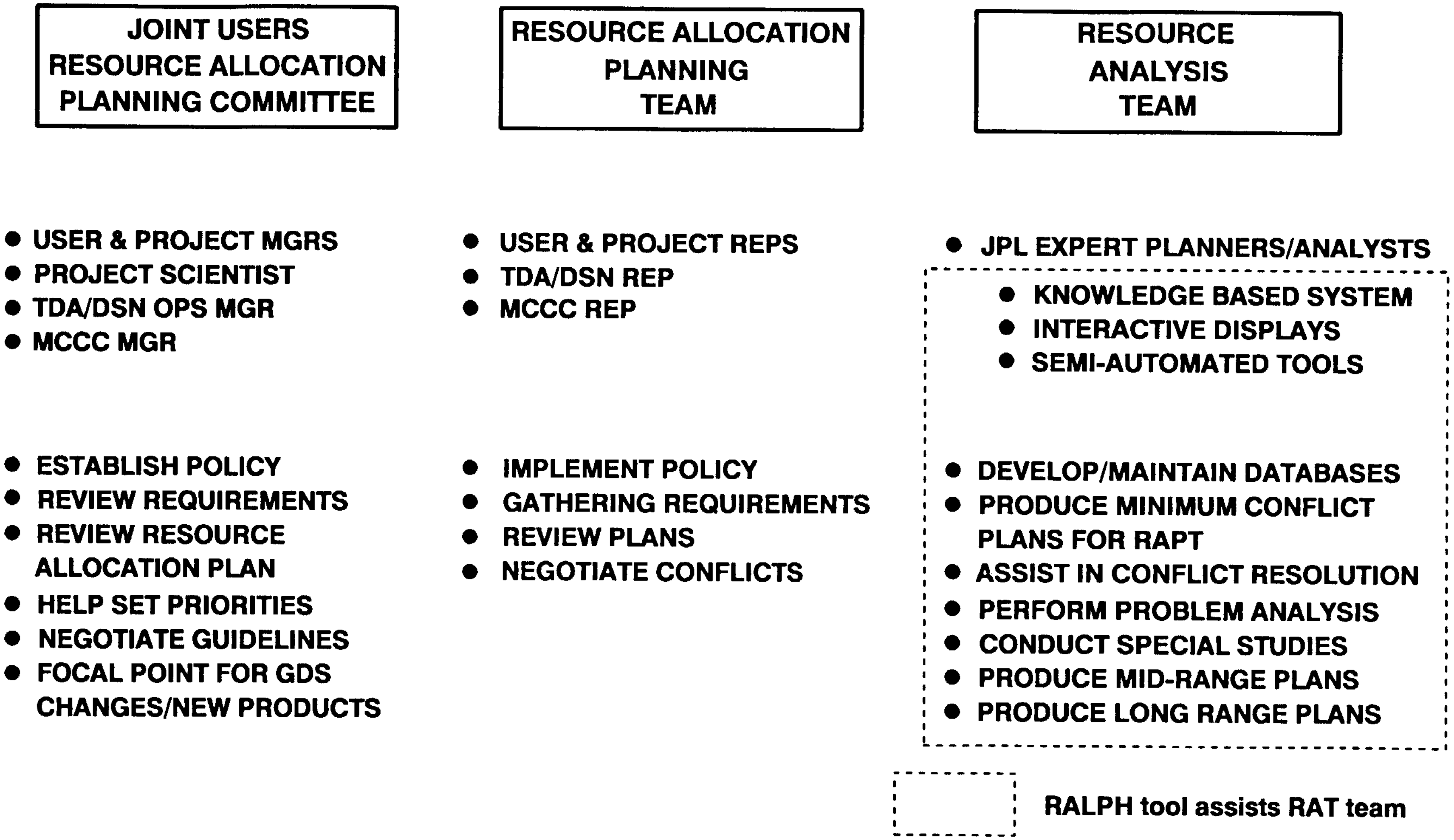
RESOURCE ANALYSIS TEAM

- JPL EXPERT PLANNERS/ANALYSTS
  - KNOWLEDGE BASED SYSTEM
  - INTERACTIVE DISPLAYS
  - SEMI-AUTOMATED TOOLS

- DEVELOP/MAINTAIN DATABASES
- PRODUCE MINIMUM CONFLICT PLANS FOR PART
- ASSIST IN CONFLICT RESOLUTION
- PERFORM PROBLEM ANALYSIS
- CONDUCT SPECIAL STUDIES
- PRODUCE MID-RANGE PLANS
- PRODUCE LONG RANGE PLANS

RALPH tool assists RAT team

Figure 1 Resource Allocation Planning Process



JOINT USERS RESOURCE ALLOCATION PLANNING COMMITTEE

- USER & PROJECT MGRS
- PROJECT SCIENTIST
- TDA/DSN OPS MGR
- MCCC MGR

- ESTABLISH POLICY
- REVIEW REQUIREMENTS
- REVIEW RESOURCE ALLOCATION PLAN
- HELP SET PRIORITIES
- NEGOTIATE GUIDELINES
- FOCAL POINT FOR GDS CHANGES/NEW PRODUCTS

RESOURCE ALLOCATION PLANNING TEAM

- USER & PROJECT REPS
- TDA/DSN REP
- MCCC REP

- IMPLEMENT POLICY
- GATHERING REQUIREMENTS
- REVIEW PLANS
- NEGOTIATE CONFLICTS

RESOURCE ANALYSIS TEAM

- JPL EXPERT PLANNERS/ANALYSTS
  - KNOWLEDGE BASED SYSTEM
  - INTERACTIVE DISPLAYS
  - SEMI-AUTOMATED TOOLS

- DEVELOP/MAINTAIN DATABASES
- PRODUCE MINIMUM CONFLICT PLANS FOR RAPT
- ASSIST IN CONFLICT RESOLUTION
- PERFORM PROBLEM ANALYSIS
- CONDUCT SPECIAL STUDIES
- PRODUCE MID-RANGE PLANS
- PRODUCE LONG RANGE PLANS

RALPH tool assists RAT team

Figure 1 Resource Allocation Planning Process

insights of benefit to those involved in the creation of similar systems.

RALPH has enabled the JPL Resource Analysis Team to extend its planning horizon from the three weeks common in the mid-1980s to ten years. In doing so, it provides a valuable planning tool for ground-based radio astronomy, mission and sequence designers of current unmanned deep space and planetary projects, and planners of future projects. Providing the ability to quickly derive answers to 'what-if' questions posed by JPL and NASA management, RALPH has proven to be a unique and worthwhile resource.

This paper presents a discussion of the transitions that have occurred during an evolutionary design and implementation cycle.

COHESIVE DESIGN TEAM

From the inception of the project in 1985, a relatively small group of developers and users have worked cooperatively toward an illusive goal. Though from different organizations within the JPL matrix, developers and users have built a working relationship based on constant personal communication. To the team, this has meant that formal weekly design meetings are reinforced by daily informal sharing of progress, frustrations, an ever-expanding user wish list, triumphs and failures.

Through the first three years of design and prototyping, purely through an accident of logistics, the teams inhabited the same building, separated only by a flight of stairs. This providential co-location was of tremendous importance to the ultimate success of the project. The development team participated in the 'hands-on' production of planning products during this period, and through these efforts were able to both confirm what they knew and to identify what they did not yet understand.

'LIVE-IN' KNOWLEDGE ENGINEERING

A fortuitous decision in the early stages of RALPH design was that to build a true expert system. As will be explained later, that decision was altered in stages as the design matured, but the effort by development to understand the Planning Methodology was already underway.

Planning is not scheduling; though the end result may well be a schedule, it requires a unique mind-set that does not come easily to some. A plan may be differentiated from a schedule by the process of its creation. Typically, a planner has much more to consider than simply how to fit some irregular pieces together to force them into a confined space. Rather, he must juggle the complexities of intertwined impacts that his decisions may have on the entities being scheduled. A prime example is that Resource Allocation Team plans (schedules) have as their principal aims: (1) the maximizing of science return from each of the spacecraft being tracked, (2) optimization of resource use, and (3) spacecraft health/survival.

The need to understand the planning mind-set became apparent during the earliest stages of the Design Team's work, and, as a result, the developers virtually 'moved in' with the users. Due to the close proximity of our offices, members of the development team had begun to spend virtually all of their time working with the planners. Though much of their activity was typical of the information gathering stages of Knowledge Engineering, their involvement with the

Resource Allocation Team was total. At least two developers have acquired the skills of apprentice planners.

The result of this extended task has been a design that is intuitive, generic and flexible. RALPH is optimized to attack some very specific problems revolving around identification and management of resource conflicts at Deep Space Network (DSN) stations, but has already shown its ability to solve problems not specifically foreseen by the Design Team. As they acquired the planning mind-set, the development team began to realize that the solution to the primary problem, if properly implemented, could be applied readily to other resource scheduling situations.

One of the most commonly applied results is the capability to do 'what if' special studies for a variety of users. To date, those activities have included an analysis of the potential impact of NASA access to a proposed Centre National d'Etudes Spatiales (CNES) 34 meter tracking station on Tahiti, an impact study on proposed DSN support for Phobos and a DSN study to examine the cost-effectiveness of acquiring some additional hardware to minimize station downtime during equipment upgrades. The Phobos study was of some interest because, as a result of its findings, Phobos Project management altered the landing date of Lander #2 and adopted a co-location scenario for the landers. The addition of RALPH support has enabled the team to produce one or two special studies each month as a complement to the regular work load.

Each of these, and other studies, have been possible because the design offers nearly unlimited flexibility that allows the planner to describe virtually any set of resource capabilities and user requirements.

A LITTLE LANGUAGE FOR DATA TREE MANIPULATION

Prototyping the necessary Requirements and Resources data management techniques had proven the value of specialized tree structures to the development team. It became apparent, though, that without resorting to the use of proprietary software with some unacceptable limits, manipulation of data objects required a great deal of coding. The decision to create a little language to manage operations within the RALPH database has proven to have been the correct path.

Implemented in C, the resulting Tree Manipulation Base Routines (TMBR) has provided sufficient power and flexibility that approximately half of the RALPH executables are written in TMBR. The remaining code is C.

The tree structures are central to the final RALPH design. Schedules are appended to the lowest levels of the Resource Capabilities trees as they are created. The text and graphics editors, printer and plotter routines and display drivers all access schedules and user requirements via TMBR commands. Schedule changes following negotiating sessions alter the data structures via TMBR.

CART BEFORE HORSE

When the Resource Allocation Team first identified the need for software support, it seemed apparent that the conditions necessitated a schedule optimizer. The planning staff had spent years putting together schedules manually, but the capability to create a final product that provided the maximum possible support for all users while minimizing negative impacts was very time-consuming.

The inability to identify and evaluate all alternatives had precluded true planning beyond three to eight weeks. Even within this time frame it was difficult, if not impossible, to react in real time to changes in user requirements (science opportunities such as unexpected solar activity or the recent super nova) and facility capabilities (last winter's inopportune loss of the 70m station at Madrid during a period of already heavy contention).

As design work was begun, it became apparent to the team that concentrating on the optimizer would be a tactical error. The resource allocation process would be better served by a tool that could build the schedule from scratch. Optimization, though guided by the planners, had always been a people process, decision making by consensus of representatives of all involved projects. Totally removing the users from the loop would be politically inadvisable.

With the realization that the planner logically had to be done first, the team once more began looking at required functionality. Focus was initially on implementing what we had learned about the planning process, but once again, it was realized that we had not reached the· illusive 'square one'. Our planners work was being driven by written requirements levied by users. The planning software would require some sort of interface through which requirements could be input.

The requirements translator, which interprets and reformats user inputs, revealed itself to be a task of a complexity equal to that of the planner. The translator must accept widely varied input in the form of user requirements and synthesize a uniform list of times, durations, antenna designations and split coverage with tracks to support uplink and downlink for distant spacecraft that the planner can overlay on a timeline. Parsing the input file and creating a common format from requirements which may be totally generic (14 hours of tracking on 34m stations during the next seven days), science priority specific, spacecraft or mission event-driven, or innumerable combinations is more time and resource consuming than creating a plan. (see Figure 2)

Another pivotal innovation that has made the system such a valuable tool was the concept of Generic Requirements.

Building a schedule is a series of controlled, rule-based reactions to requirements imposed by the participants. Each scheduling exercise calls for thousands of individual decisions and has thousands of potential solutions. As in a game, the complexity increases geometrically with the number of rules. If projects could be convinced that there were advantages to loosening or reformatting their tracking requirements, or reducing the number of rules in the game, the planner, whether man or machine, would have many more options and could ultimately create a better plan.

That campaign has been won. RALPH was implemented with the ability to interpret inputs and build plans whether driven by specific or generic requirements, but the user projects, realizing the positive impact of generic requirements use them for all tracking except that supporting project-critical events such as encounters and maneuvers.

The goal, then, was to create a requirements translator/planner/optimizer to provide the conflict resolution process a solid starting point. This is not to minimize the quality of RALPH plans for, starting with the highest priority requirements, the planner derives from

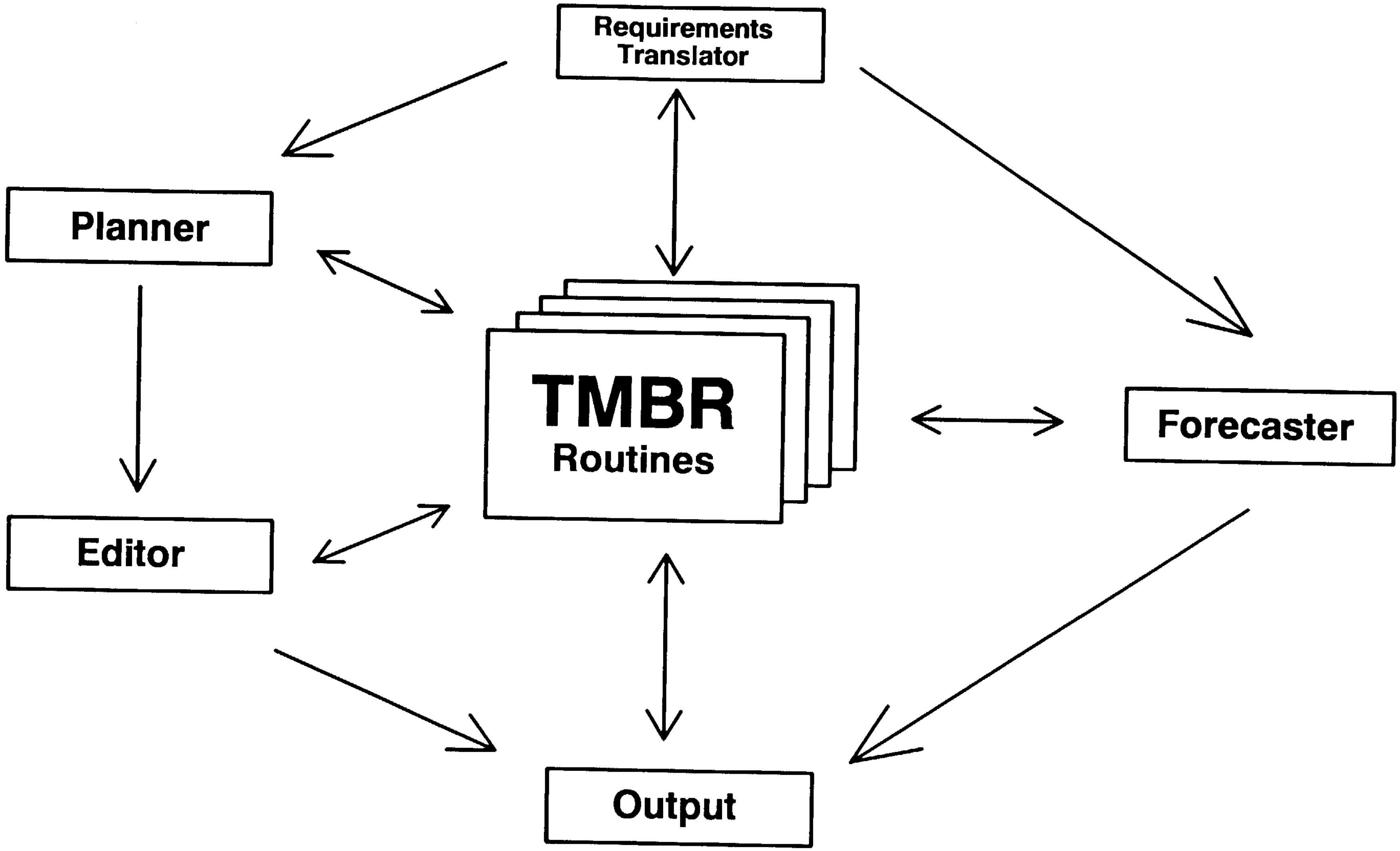


Figure 2 RALPH Architecture

the multitude of possibilities the 'best' option. And, though the planner can be forced to produce a conflict-free product, as we have previously discussed, we believe that the committee of project representatives should resolve the most difficult conflicts though consensus decisions.

HOW FAR SHOULD AUTOMATION GO?

When the RALPH tool kit was originally conceived, one of the ultimate goals was to determine the appropriate mix of decision making by software and by humans. However, resource allocation is a very complex function the result of which ultimately determines or, at least strongly influences, the volume and mix of science and engineering data to be returned from each spacecraft. The science investigation teams for each project could make a valid case for increasing the amount of coverage granted their spacecraft or ground-based activity. For these reasons, and for those we have discussed elsewhere, the final steps in making tracking decisions is, and will continue to be, made by a team of project representatives working with Resource Analysis Team planners.

Human nature, then, is an undeniable obstacle standing in the way of a totally automated planner. With the adoption of generic requirements and with RALPH's event priority logic delivered, there is no technical reason that a set of rules could not be assembled that would allow the software to create a conflict-free schedule.

If total automation were a goal, a second impediment, one that would have to be overcome by careful design, would be the need for a comprehensive rule set defining the relationship and priorities between every combination of supported projects and a second set defining every potential contingency that might alter the requirements of each project. In planning, decision making must take into account the present situation, but must often also consider inter-project trade-offs used recently in granting (or withholding) support.

A planner may, for example, deny Voyager 1 several hours of tracking time during a specific week in favor of ICE, but often does so with at least the informal understanding that that time will be 'repaid' when the total tracking situation allows it. Any such trade-off, including the intention to do a payback, is always driven by the ultimate goal of maximizing science return for all supported projects. Any fully automated system would have to be capable of similar decision making to be acceptable.

START-UP LAG

Valid statistics demonstrating the cost-effectiveness of the RALPH development effort have been compiled and advertised. At the same time, we have recently been overwhelmed by more than normal negotiating time, one of the specific problems that RALPH was built to minimize. We have been able to demonstrate that both the process and the tool are valid and are functioning as designed. We are, however, victims of uncontrollable past circumstances.

Had the capability to do long-range planning existed in the early 1980s, JPL and NASA would have had a tool to permit optimized re-scheduling of payloads when shuttle flights resumed following the post-Challenger hiatus. Long-term planning completed before the Challenger accident had scheduled most upcoming launches, encounters and other tracking-intensive events to minimize serious conflicts.

With the resumption of STS-based planetary launches, however, the schedule for the late 1980s and early 1990s has become anything but optimal. Magellan (MGN) and Galileo (GLL) were launched six months apart with mission designs vastly different than those originally planned. As a result, both are headed for Venus and periodically have very similar view periods (that is, occupy the same part of the sky from an Earth perspective). Sharing significant parts of the same sector of the sky are two of the Pioneer spacecraft. As GLL approaches Venus for a fly-by and concurrent trajectory correction, planners are faced with providing Galileo nearly continual coverage while providing at least survival coverage for MGN and the Pioneers. This is just the situation planning is designed to avoid.

Software projects implemented to correct or improve an on-going situation should provide some strategy to go back and correct the short-comings of the past. The danger is that sponsor confidence can be badly damaged unless the development organization is able to foresee start-up deficiencies and make them known to management.

The RALPH Design Team had anticipated this 'lag' to some extent and, consequently had taken steps to prepare management. Two additional problems prevented a totally adequate reaction to the coming situation. First, when launches resumed, there was not sufficient lead time to allow flight projects and mission planners to react properly. In addition, these events occurred at a time when Resource Allocation Plans were new to project management and the plan's credibility had to be established before any reaction could be mounted.

When plans with enormous levels of antenna contention began to appear, the negotiating process reacted by slowing down from the excessive work load. It appeared at first that neither the software nor the resource allocation process were working when, in reality, the quality of both was absolutely valid under the circumstances. What was required was a doubling and re-doubling of negotiating time until the most difficult periods had been freed of conflicts.

COST EFFECTIVENESS

While Resource Analysis Team benefits of the RALPH tool kit and planning products produced through its use are easily measured, a significant portion of its positive impact per dollar invested cannot be estimated with any degree of accuracy. The latter is largely because RALPH provides long-range planning capabilities not available in the past from any source. The semi-annual mid and long-range plans as well as special studies, done largely by special request and with quick turn-around, have become working tools for near term decision making and long-range planning by JPL management and NASA Headquarters.

A significant side benefit of RALPH is its support of RAT team activities including data entry, plan generation, and conflict resolution meeting support. The 1988 estimated savings was well over 7000 person-hours a year. A significant part of this is the fact that preparation of weekly plans historically required 25 hours of an experienced planner's time. Now, a RALPH planner with far less experience can produce a plan in five hours, most of which is consumed by data entry.

The editors and plotting routines make possible the production of diagnostic tools never before available. The majority of the 7000 hours, though, is reflected in the fact that the mature state of RALPH allocation plans has reduced the number of negotiating sessions to only one weekly from the former three or four. This reduction is worth well over 6000 hours yearly. The RALPH four year development costs have been about $3 million.

HYBRID ARCHITECTURE

As a design incorporating heavy reliance on data tree structures emerged, the designers realized that any attempt to continue to rely wholly on the original expert system concept was invalid. Rather, conventional algorithmic structures could be used for much of the planning as well the supporting editors, input/output and interpretive modules.

The RALPH design continues to rely on rule-based decision making for such tasks as creating best fit schedules and, at a more detailed level, making support decisions based on sets of contention variables. Each of the TMBR modules, then, has been designed either as a rule-based routine or as a traditional algorithm.

PAINFUL TRANSITIONS

To have the opportunity to even begin an innovative task requires equal doses of optimism and masochism, innovation and conservatism. In a large organization, it also requires management with the foresight to charge an enemy hidden in the mists of uncertainty. Innovation can make you seem a genius or a fool. Innovation is an arena for those who understand that not every attempt is a win, but who believe that the goal is worthy and the aspirants are equal to the test.

RALPH development has been treated somewhat differently than many software prototypes. Whereas the prototyping environment is most often laboratory-like, isolated from the atmosphere of real world production, RALPH has been an 'on-line prototype' from its first delivery in 1986. We have found this to be an optimal state offering the users the latest available technology while at the same time allowing quick turn-around when delivering new features or bug fixes.

Daily use of advanced prototypes also offers the developer a realistic perspective of the true usefulness of his creation. Isolated prototyping, if not carefully designed, can mask the behavior of subject software under the hands of perhaps less sophisticated users who can usually be counted on to do the unexpected in the course of meeting daily production goals.

Throughout the prototyping period, the status of the software, and of its relationship to the process it supported, was under constant peer review. That feedback continues today through both formal and informal feedback from the user community and the RALPH review board.

During four years of on-line prototyping, current configurations on development and production machines had been managed by knowledgeable members of the design team. The sponsoring organization, JPL's Flight Project Support Office (FPSO), determined that RALPH had achieved a state of maturity that demanded the end of informal deliveries and consequent introduction of formal configuration management (CM).

Though neither users nor developers foresaw this as a problem, we learned over the next two or three months that two distinct philosophies existed. The design team assumed an indeterminate transition period controlled largely by the relative ease of the version 4.0 delivery. This delivery followed a major rewrite during the transition from RALPH's early format to the previously described TMBR/C version, a non-trivial metamorphosis. Configuration Management had assumed a transition day to occur before delivery to test. The acceptance test period was extensive and required several re-deliveries. Ultimately, it was evident to all that a transitional period had occurred naturally and that formal, third party CM was applicable only after true stability of the final prototype version had been achieved.

WHERE WILL IT END?

References have been made to the mid-course corrections that have been necessary to refine the design team's targeting. The result is that, to date, a tool set with enormous positive impact has been created for the users. Through use of the growing set of tools, both the Resources Analysis Team and the projects they support have conceived an extensive list of additional capabilities that would expand the team's abilities even further.

Not the least of these is the suggestion that, in the coming era of growing international cooperation and sharing of resources,[2] the JPL resource allocation process and the RALPH tool kit would be an ideal means to provide optimized planning and scheduling for the world space science community. The Phobos and CNES studies mentioned earlier, as well as support for such multinational efforts as Ulysses, Giotto and Galileo, have provided valuable experience in international cooperation.

CONCLUSION

The RALPH experience has emphasized a number of lessons which, while not unique to this effort, will have been of benefit as we embark on future projects. Tantamount to a secure development environment is a sponsor who understands the risks and potential benefits and is willing to support development through both good and bad times. During development, success of a unique system is highly dependent on a close and lasting relationship between developers and users. Such a circumstance is a virtual guarantee that delivery will be free of misgivings and disappointment.

An expert system is traditionally design to *replace* an expert. The goal of RALPH implementation has been to relieve the 'front end' burden of tedious scheduling from the expert planner and to move him to the 'back end' of the system where his expertise can be concentrated on analysis and on providing recommendations prior to negotiations.

In the creation of a precedent-setting system, it is vital for the developers to take unusual strides to be assured that they understand the full implications of the user's requirements. The system initially conceived and requested by users may not be the solution to his problems, for, as we have experienced, the methodology may change in response to the power of the tool. Proceed with caution, be flexible and schedule deliveries that assume change. Regardless of the care given to the original design decisions, significant changes are likely when the methods chosen break new ground. And, finally, plan a lengthy transition from prototyping to a more formal, controlled environment.

REFERENCES

1. Berner, C. A., R. Durham and N. B. Reilly, "Ground Data System Resource Allocation Process", 1989 Goddard Conference on Space Applications of Artificial Intelligence, May, 1989

2. Dumas, L. N., G. A. Briggs, M. S. Reid, and J. G. Smith, "Opportunities for International Collaboration in Deep Space Communications and Tracking", Proceedings of the Second Annual AIAA/JPL International Conference on Solar System Exploration, Pasadena, August 1989

N90-22297

THE ROLE OF ARTIFICIAL INTELLIGENCE TECHNIQUES IN SCHEDULING SYSTEMS

Amy L. Geoffroy, Daniel L. Britt & John R. Gohring
Martin Marietta Information Systems Group
Denver, CO.

ABSTRACT

Artificial Intelligence techniques provide good solutions for many of the problems characteristic of scheduling applications. However, scheduling is a large, complex heterogeneous problem. Different applications will require different solutions. Any individual application will require the use of a variety of techniques, including both AI and conventional software methods. The operational context of the scheduling system will also play a large role in design considerations. The key is to identify those places where a specific AI technique is in fact the preferable solution, and to integrate that technique into the overall architecture.

Introduction

As Artificial Intelligence (AI) techniques have moved from the laboratory into complex applications, two things have become apparent. First, frequently more than one AI technique is required to satisfy the multitude of requirements in a large, complex operational domain. Second, AI techniques alone are either insufficient or not the most efficient means of performing these complex tasks - they must be integrated with standard software (and sometimes hardware). Scheduling in resource constrained domains is one example of this type of problem.

Resource constrained scheduling[1] is a heterogeneous problem in two ways. First, there can be tremendous differences between the characteristics of various applications' scheduling problems, even within space applications (e.g. ground processing vs. on-board experiment scheduling). Because of these differences in requirements, successful scheduling solutions for these applications will generally be somewhat different for each application. Second, there are multiple requirements within a single application. In experiment scheduling, for instance, there are requirements to limit search through the space of all possible schedules, and also to represent and manipulate quite complex resources. These are distinct subproblems, and techniques used to limit search and those used to handle complex resources will be different. Any given scheduling application may have a large number of distinct subproblems.

Scheduling within an operational context consists of both a core scheduling problem, and the problem of the scheduler's place in the operations context. The core problem

[1] Scheduling problems which are not resource limited are very different in character from resource constrained scheduling and are not discussed here. The scheduling of space-based activities is resource limited, as are most ground-based operations. For the remainder of the paper the term scheduling is used to refer only to resource constrained scheduling.

will be determined by certain formal characteristics of the scheduling domain. These include considerations such as the computational complexity of the domain (as determined by the number of items to be scheduled, the structure of the items to be scheduled, the granularity of the schedule, etc.), and the predictability of the domain. The operational context drives a host of additional requirements, such as distribution of control over scheduling, human-machine interfaces, and the role of the scheduler in contingency handling.

Satisfying scheduling requirements within an operational context calls for careful examination of the specific application, and identification of the place of AI and standard techniques within an overall system architecture that address the particular characteristics of the application.

The first section of this paper introduces the resource constrained scheduling problem. The second section provides a general discussion of how AI techniques are well suited to solving many scheduling subproblems. The third section shows how the particular characteristics of specific scheduling problems will determine the appropriateness of different solution methods, with particular attention paid to AI methods. Included in this section are examples of how different applications may differ in their requirements, how a single application may consist of many subproblems, and how AI, Operations Research (OR), and other conventional software techniques may be profitably combined to provide solutions to an application's scheduling problem. The fourth section discusses ways in which the operational context within which the scheduler will reside drives many design considerations. In the concluding section we summarize the discussion and consider how design for AI systems fit into the overall design of large, complex systems.

Scheduling in Resource Constrained Environments

Resource constrained scheduling is the fixing of activities on a timeline such that those activities may be performed at the time specified by the schedule. This entails the coordination of requisite resources, the availabilities of required ambient conditions, and the interleaving of activities which compete for resources.

Certain prerequisites must be met in order for scheduling to be feasible. Both resource availabilities and the activities' requirements must be roughly predictable. All relevant characteristics of activities and resources must be expressible. To the extent that these requirements are met, predictive scheduling - scheduling for a future time period - can be successfully performed.

For this paper, we will define three levels of description for activities: objectives, activities, and subtasks. An objective is the goal or purpose of the task, e.g. to produce a crystal. An activity is one well-specified way of fulfilling an objective, e.g. one run of the crystal growth experiment. A subtask is a step that is a part of the performance of an activity, e.g. the set-up phase in the crystal growth experiment.

TARGETING EXPERIMENT

Resource requirements:

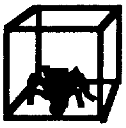
	Pointing	Calibration	Data Collect	Shut Down
Power	50	150	190	50
Heat Rej	50	150	190	50

Conditions: Target must be available for data collection phase
Vibration must be < x for calibration and data collection
Atmospheric pollution must be < y

Side effects: Creates vibration when rotating in pointing phase

Time windows: Mon. - Fri. 11am - 6 pm (when ground support is available)

SPIDER EXPERIMENT

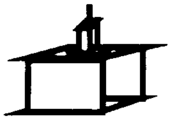


Resource requirements: Continuous power & heat rejection, 25w each
1 Crewmember (PO or PS) for observation phases

Conditions: Observation phases vibration < z
Observations must occur at least 10 minutes after "sunrise", during "daylight" only

Coordination: Observation phase of this experiment should co-occur with the filming phase of the Public Relations filming

CRYSTAL SAMPLE GROWTH EXPERIMENT



Resource requirements:

	Set-up	Heat	Grow	Centrifuge	Analysis
Crew	1	0	0	1	1
Power	0	500	250	110	25
Heat Rej	0	200	500	110	25

Conditions: Sample growth < p vibration
Sample analysis < q vibration

Gasses and liquids: 20l Helium 5l Argon 2l H 0

Side effects: Centrifuging induces X vibration

PUBLIC RELATIONS FILMING

Resource requirements:

	Set-up	Film	Break-down
Crew	1	1	1
Video Cam	1	1	1
Power	0	125	0
		1 roll film	

Conditions: No venting during filming
Filming <x vibration

Coordination: Filming phase must co-occur with spider obs

Figure 1. A simplified example of activities which might require scheduling for Space Lab, described in terms of their requirements.

An example will be used to illustrate some characteristic problems commonly found in scheduling applications. This is not intended to represent a "generic" scheduling problem. As we shall argue later, applications vary enormously in the ways their scheduling problems may be characterized, and these differences preclude the possibility of a single representative scheduling problem. Rather, the example serves to illustrate one typical version of a scheduling problem.

Fig.1 shows a simplified example of a hypothetical scheduling problem. Suppose that three experiments - one using a targeting instrument, another involving a crew member's observation of the activity of some spiders, and another involving the generation and centrifuging of some samples - and a public relations filming activity, all require scheduling for Space Lab .

For each of these activities certain resource requirements must be met: power (for the targeting and sample experiments), crew time (for the spider and sample experiments), etc. Insofar as these resources are limited, different activities may compete for these resources, requiring coordination of the activities. Conditions for each of the experiments also must obtain - the targeting instrument must be able to acquire its target, and may require a minimum vibration, while the centrifuge phase of the sample experiment may generate a certain amount of vibration. Additionally, there may be a requirement to film a crew member performing the observations in the spider experiment for a publicity film, and this will require coordination of the filming's timing, resource and conditions requirements with those of the spider experiment. Because time and resources in space are rare and costly it is of the utmost importance to generate as efficient a schedule as possible. The process of producing an efficient schedule which provides the necessary coordination of resources and conditions for these activities can be quite complex.

There are several sources of difficulty in this scheduling problem which are found in many different scheduling applications. The scheduling objects, i.e. the activities and resources, can be quite complicated. The relations between these objects can also be complicated. Ensuring that these objects and their relationships are all appropriately represented, and that there are means of reasoning about them which will allow even the verification of schedule validity is no trivial task. However, the most critical challenge is controlling the combinatorics of the problem - producing a good schedule given all the possible schedules that might be generated.

Combinatorics. It has been demonstrated that procedures which guarantee an optimum solution to scheduling problems are either NP-complete or NP-hard (Ibaraki & Katoh, 1988), depending on the characteristics of the particular problem. Realistic scheduling problems are most frequently NP-hard. What this means is that the computation time for finding an optimal solution to a scheduling problem increases exponentially or worse as a function of the number of subtasks and resource disjunctions (choices between unique resources in a resource pool which satisfy the resource requirement) under consideration.

In the example given in Fig. 1 above, each of the phases of each experiment counts as one subtask (e.g. the targeting experiment consists of four subtasks - pointing, calibration, data collection, and shut down) yielding 14 subtasks. One three-valued resource disjunction is represented in the choice between crewmember types. For any given subtask requiring crew time, you may use any one of three types of crewmembers - Payload Operators (PO's), Payload Specialists (PS's), or Mission Specialists (MS's), unless some restriction is specified. Multiplying out each subtask by the resource disjunctions found in each subtask, we find that n=27. This, however, represents only a small subset of a realistic scheduling problem for the Space Lab application. In a real application, there are many more experiments under consideration, each experiment consists of more subtasks, there are more resources and there are more disjunctions of those resources .

Parunak (1987) states, "... consider a problem that in the worst case requires 2^n microseconds to solve, where n is the size of the problem .... [a] problem of size 10 will require no more than .001 seconds to solve... of size 40... as long as 12.7 days ... of size 60, 366 centuries! ... Even with a thousand fold increase in speed, the size of problem that we could guarantee to complete in 366 centuries increases only to 70."

These increases in computation time are a direct reflection of the increasing size of the space of possible schedules - all possible combinations of placements of subtasks on the schedule. Most realistic, complex scheduling problems are not amenable to optimization techniques simply because of the combinatorics involved. Thus, methods must be derived to arrive at solutions without searching the entire space of all possible schedules.[2]

The Suitability of AI Techniques for Scheduling Problems

There are a number of characteristics of scheduling problems that make them amenable to AI solutions, some having to do with the combinatorics problem, others having to do with other aspects of scheduling - requirements for rich representation techniques, exploitation of constraints, planning problems, and the utility of expert knowledge.

Combinatorics. One of the main goals in the development of AI as a field has been to devise methods which

[2] An alternative approach to this problem has been developed in Operations Research (OR). There has been a great deal of work in OR on algorithmic methods which yield good, but not optimal solutions to scheduling problems, but these are also limited, either by size of the problem that can can reasonably be attacked by these methods, or by the limiting assumptions that must be made for the methods to be effective (Graham, 1978). Also, these techniques require an "objective function" - a formula which supports precise measurement of schedule value which the algorithm tries to minimize or maximize. In many real scheduling applications, there is no objective function. Human schedulers cannot formulate a precise mathematical combination of their many often conflicting goals (e.g. resource efficiency, relative priorities for activities, fairness) that represents the value of a schedule. However, these methods may prove quite valuable if used on small subproblems for local decision making.

circumvent the problems of combinatorics by intelligently guiding search through enormous spaces, and these methods are applicable to scheduling problems. (In fact , a number of these methods have been adopted by OR practitioners and others, and are now considered standard computer science techniques).

Representations. Many of the objects in these domains may be considered semantically rich, hierarchically or heterarchically structured, with many different types of characteristics (consider the description of the Space Lab experiments given above). The representation techniques developed for AI, particularly object-oriented programming, are ideally suited for this domain.

Constraints. There are many constraints that must be respected, or which can be exploited in developing a schedule, and constraint propagation and relaxation techniques have been well developed in the field.

Planning. Some aspects of the scheduling problem turn out to be problems in planning - in some cases an objective may be specified, but the actual activity to support that objective may be underspecified. The unique specification of the activities which will achieve the objective is a planning problem, another area in the AI field.

Expertise. Much of the know-how in scheduling is the purview of human experts, and may be amenable to expert system techniques.

There are other characteristics of scheduling which make it a good candidate for AI techniques, but these examples suffice to show that AI is a good path to explore in creating solutions.

Matching Techniques to Scheduling Problem Characteristics

The discussion above provides a general picture of how AI might be applied to scheduling problems, but little specific about how to actually apply these techniques. This is because scheduling is a heterogeneous problem. The exact form of a solution to any given scheduling problem depends on the characteristics of the particular problem. A sample of relevant characteristics are:

I) Activities
  A) Total number
  B) Complexity
    i) number of subtasks
    ii) number of resources/subtask
    iii) dependencies between
      resource requirements
  C) Similarity of activities
  D) Fixedness
    i) Timing
    ii) Number of different ways of
      achieving the same objective
  E) Fragmentability
  F) Co-dependencies between
    activities or subtasks

II) Resources
  A) Number
  B) Disjunctions
  C) Complexity
  D) Similarity of resources
  E) Co-dependencies between
    resources

III) Time
  A) Granularity
  B) Schedule duration relative to
    activities' and subtasks' durations

IV) Schedule
  A) Repetitiveness
  B) Resource costs
  C) Activities' values
  D) Goals
    i) explicitness
    ii) number
    iii) types
    iv) variety

V) Methods used in current operations
  A) Adequacies/Inadequacies
  B) Experts
    i) existence
    ii) quality of performance

This list is not intended to be an exhaustive catalog of all relevant factors, but it represents many important ones. For instance, the number of subtasks (IBi) and the number of resource disjunctions (IIB) will determine the size of the combinatorics problem. The complexity of activities (IB) and resources (IIC) will determine the complexity of the representations used. Number of ways of achieving the same goal (IDii) indicates the extent to which a planning problem exists.

The values of these characteristics can vary considerably between different scheduling applications. The specific characteristics of a given scheduling application will determine what combination of techniques, both conventional software and AI methods, are most appropriate. There are three major points to be made here. First, different applications will demand different solutions. Second, because real scheduling applications are really a combination of a number of thorny problems, a single application will probably require more than one methodology. Third, these methodologies may often be a combination of AI and conventional software methods.

Differences between diverse applications' characteristics will demand different solutions. A few examples serve to show how the different characteristics of a problem can help to determine appropriate solution methods. For instance, in domains where there are many co-dependencies between activities, between subtasks or between resources, constraint propagation techniques will be extremely important. Search methods that evaluate current state as a function of schedule "goodness" so far and/or projected distance to goal (e.g. best-first search, genetic algorithms) work well in domains where a) scheduling goals are explicit, b) their values may be defined quantitatively, c) a function defining the combination of these values may be defined to reflect an overall schedule value, and d) these values are easy to measure on an existing schedule, but not otherwise. Where human experts perform the scheduling function quite well, an expert systems approach would generally work well, but not in domains where human scheduling is considered inadequate.

An interesting comparison of matching techniques and domain characteristics is shown in contrasting scheduling for some Deep Space Network (DSN) problems and scheduling the Laboratory experiments onboard Space Station. The DSN problem can be characterized as having an extremely large number of activities, each fairly simple - consisting of one or few component subtasks, and requiring a small number of resources. Also, the activities to be scheduled are very similar to one another, using basically the same resources in basically the same ways. In contrast, there are fewer activities to be scheduled for the Space Station Laboratory, but each of the activities is more complex. Each activity consists of a larger number of subtasks, each requires a large number of resources, and the activities are much less similar to each other.

A scheduling strategy adopted for the DSN problem might be to create an initial schedule paying little attention to resource limitations, and in which resources are overbooked, then to shuffle activities on the schedule to try and ameliorate the overbooking. The primary strategies in use here are backtracking and evaluation of entire candidate schedules. In contrast, for the Space Station Laboratory problem, a scheduling system might use constraint propagation techniques and a number of intelligent heuristics to create an initial schedule which is conflict-free. These methods would concentrate on finding places on the schedule where each activity fits without resource or other constraint violations. The methods used here would be constraint propagation and local optimization. These two different approaches each work well for their intended applications.

Consider trading the approaches between the applications. To fully appreciate the implications of this trade, it helps to consider the characteristics of the different domains more abstractly. Imagine activities to be n-dimensional shapes that must be packed as tightly as possible. One dimension represents time, and each of the other dimensions represents a resource required for the activity. The space into which the activities must be packed will have the same number of dimensions as the total number of unique resources used by all the activities, plus one dimension for time. Figure 2 shows a representative simple case of a four-dimensional task.

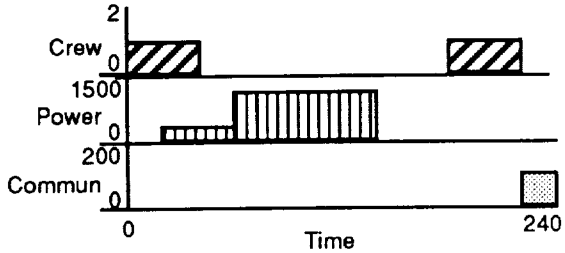


Figure 2. A task requiring three resources, crew time, power, and communications time, against time. This translates into a four-dimensional shape for the task.

Because the DSN activities are fairly simple and similar, the shapes for each of the tasks will be much alike. Because there are few resources under consideration, the n-dimensional space into which they will be packed has a small number of dimensions. In contrast, the Laboratory tasks are much more complex and less similar, so the task shapes will be very different from each other. The n-dimensional space into which they must be fit has a large n (over 200 resources are used in scheduling for one version of this problem).

The approach of initial random placement followed by shuffling to ameliorate overbooking is good for cases where the shapes are basically similar, because the shapes representing each task are pretty much interchangeable. Random placement and shuffling are dreadful strategies, however, when the shapes of activities are very different, and there are a large number of dimensions to be matched. This is because (except in a resource-rich environment) random placement is unlikely to yield viable placements for complex activities. Further, randomly substituting an activity which is currently an unsuccessful fit for one that is a successful fit only works insofar as the successful item is blocking the resources needed for the other item. The more dimensions under consideration, and the more different the two activities are on those dimensions, the greater the probability of failure.

The approach which focuses on goodness of fit for activities at each step in schedule generation is good for scheduling dissimilarly shaped activities in a large number of dimensions because it focuses on finding such fits. However, it is computational overkill to use this approach for shapes which are highly similar and are to be fit in a small number of dimensions. Most of the decision making strategies used in creating an initial schedule would be irrelevant in the DSN application. Also, since this approach does not support random modifications (a good idea for DSN-type tasks), it fails to take advantage of some simple strategies that, for the DSN domain are effective at reaching better quality schedules quickly.

Different methods will be used for any single application. Any given scheduling application may consist of a combination of hard problems. Some of these problems may interact, so that solutions must be co-designed, while others may be independent, and the design for solving the different problems may be performed independently.

The example used in Fig. 1 illustrates a few of these types of problems. There are several different kinds of constraints that must be respected in order for these experiments to be successfully scheduled. Two of the major constraints are resource requirements (e.g. power and crew requirements) and temporal co-dependencies (e.g. do the public relations filming while an astronaut is observing the spider experiment). These two constraint types require different aspects in representation, and different computational methods to ensure that these constraints are met. However, both constraints are used to compute answers for a single problem - where an activity may be placed that meets all of its constraints. Because of this, their design must be tightly linked.

In contrast, there are other difficult aspects of this scheduling problem that are fairly independent of the computations for temporal and resource constraints. The management of several of the resources in this scenario is complex. Crew, for instance, has certain restrictions not only on total amount of time worked in a shift, but on the combination of experiment types that may be successively scheduled, preferences for different activities, relative experience with and qualifications for different activities, etc. Data transmission is a complex combination of real-time transmission and storage with subsequent transmission, where transmission times may be determined by what items appear on the final schedule. These resources require complex management systems themselves, which may take on a variety of forms (e.g. expert systems, OR methods, etc.) depending on the details of the management problem. The design of each resource manager, however, is nearly independent of the constraint propagation methods for temporal and resource constraints - the only requirement is that the manager provide the type of information about the resource required for scheduling.

The best solutions may combine AI and more conventional software techniques. Because any scheduling application consists of a variety of problems, it will generally be the case that some of the solutions will consist of conventional software methods, and some will use AI techniques.

Many of the examples given above have illustrated the utility of AI solutions to problems in various applications. The most obvious example of an area for conventional software are parts of the scheduling process which are straightforward and algorithmic. Most applications, for instance, require a good deal of bookkeeping - tracking what resources and conditions are available where on the scheduling timeline, and updating those availabilities as the scheduling process proceeds. This is best suited to conventional computing procedures.

There are also interesting possibilities for combining conventional and AI techniques to attack the same problem. Several recent systems (e.g. Berner, Durham & Reilly, 1989) have combined AI and OR techniques. It is possible, for instance, to use a heuristic method to decompose the scheduling problem into subproblems which are more tractable, and then use OR or optimization techniques to solve the subproblem (Britt, Geoffroy & Gohring, 1990). It also might be promising to extend the multi-perspective scheduling strategies used in OPIS (Smith, Fox and Ow, 1986) to a multi-technique strategy, where based on the current state of the problem (e.g. whether resource bottlenecks appear, how large the current search space is, etc.) different scheduling techniques might be selected, and these techniques might be a variety of AI and OR techniques.

The Operational Context

The scheduling problems described above are realistic, and solutions to those problems can be embedded into real operations. The scheduling problem as discussed so far may be considered the core scheduling problem. Planning and scheduling in the operational context, however, entails much more than just the core scheduling problem, and the larger operational context complicates the requirements for a scheduling system. Some of these additional requirements may be addressed by systems which are entirely separate from the scheduling system. So for instance, some kind of support to help users formulate activity definitions will be required. A separate activity editor can be implemented to fulfill this requirement. Such an editor would be a good candidate for intelligent human-machine interface techniques. The design of the core scheduler will remain untouched by the requirements for the activity editor. However, there are other requirements levied by the operational context that must be refelcted in the design of the core scheduler itself.

Three major complicating factors will be discussed here. First, the scheduler performs a function which must be integrated with the other functions of the larger planning and scheduling system. Second, in the operational world, nothing ever really goes exactly as planned - there are always contingencies popping up which invalidate a schedule and require some reaction to get the schedule back on track. Third, the planning and scheduling process changes in character over time, from long-range planning to the scheduling of today's activities.

Scheduler integration. In many operational contexts, a scheduling system will reside as one node in a complex network of hardware and software systems and human users.[3] The scheduler will be involved in numerous information exchanges, e.g. receiving data about resource availabilities, or transmitting data about activities' timings. Ensuring that the design of the scheduler can support such exchanges may be time consuming, but will have little impact on the technical approach to the core scheduling problem.

However, other aspects of interactions with the other nodes in this network may drive some design decisions related to the core scheduling system. Two main issues here are control of the scheduling process, and the degree to which the actions of the scheduling system can be understood.

It is unlikely that a scheduling system for a complex scenario will always be run entirely autonomously. Even if it is feasible, full autonomy is probably not desirable. Human operations managers will want to have the final authority on decisions regarding the schedule, even if this authority is only rarely exercised. Operators must be able to control and interact with the scheduler. This places demands not only on the design of the user interface, but on the control structures of the scheduling system, particularly if the goal is a user-determined level of autonomy. It also requires that the system operate in such a way that people can understand what the system is doing.[4]

Control of the scheduler may not be limited to interactions with a single user or single user type. The scheduling of unmanned platforms such as the Earth Observation System (EOS) for instance, involves a network of science users, platform managers, instrument managers, and communications (TDRSS) managers, and all are involved in the scheduling process. Each has a different realm of authority, and each can control the scheduler in different ways. The scientists interact with the scheduler in terms of their experiments, while the scheduling concerns of the platform manager are to provide platform resources, such as power, to support the schedule, and to schedule activities which maintain the health and safety of the platform. These users interact with the scheduler asynchronously, which has implications for both the control structure and the scheduling strategies of the system. Because users interact with the scheduler

[3] There are a number of functions (other nodes) in this larger planning and scheduling context which are good candidates for AI applications, but space limitations preclude pursuing this further here.

[4] This does not necessarily require that the system reason like an expert - people can understand, for instance, that a thermodynamic model is used, even though they don't create schedules manually using that method.

asynchronously across long periods of time, the information on which scheduling is based is always in a state of flux. The scheduling strategies of the system in this scenario have to reflect the tentative state of the scheduling information available at any given point in time.

Contingencies. Schedules are based on assumptions made about activities' requirements, the resources that will be available for the scheduling period, conditions which will be true, etc. After a schedule has been generated, some of these assumptions will turn out to be false. There may be a power failure, some equipment might break, an activity might take longer than projected, or some new, previously unscheduled activity may need to be forced on the schedule. In most of these cases, the schedule will be invalidated.

The design of the scheduling system should support the ability to reformulate the schedule based on the new information. In general, it is not desirable to generate an entirely new schedule, but to repair the existing schedule, so techniques which modify existing schedules are required. Because many of these contingencies will happen during scheduling execution, reactive rescheduling must be fast. If the same techniques are used for scheduling and rescheduling, then the scheduling strategies must be designed with the speed issue in mind. If different systems are used, then careful attention must be paid to ensure that the output of the initial scheduling process can support the input requirements for the rescheduling system.

The planning and scheduling process across time. Mission operations planning and scheduling for applications such as the Shuttle or Space Station Freedom begin years ahead of the actual mission. The different phases of this process, from the strategic planning, years ahead of time; to the short term scheduling, just shortly before execution; to near real time reactive rescheduling, have different characteristics, and different requirements. The length and granularity of the timeline are different for each of these phases, and the level of detail about the activities to be placed on a timeline are also different.

This is actually an expansion of the point made earlier, that any given scheduling application has a multitude of requirements. In this case, entirely different systems may be needed to accommodate each phase of the planning and scheduling process. In addition to creating systems for each of these phases, ways of transitioning information between the phases is required. This may require some consideration of commonalities for data structures between the system, and may require that the strategies used in each phase are complementary or synchronized.

Conclusions

What can AI do for scheduling applications? As pieces of an overall system architecture, AI techniques can be used quite successfully. Methods for handling various types of computational complexity have been well explored. Techniques for representing complex objects and relationships, constraint management, planning, and representation of expert knowledge and methods are all areas of strength in the field of AI. Intelligent human-machine interface techniques can be used profitably to help systems fit into operational contexts. The key is to identify those places where a specific AI technique is in fact the preferable solution, and to integrate that technique into the overall architecture.

Much of the discussion in this paper addresses issues that are not unique to the development of AI systems, but which are pertinent to any software development for a complex functional system. The points about matching techniques to problem characteristics and taking into consideration the operational context of the scheduling application have to do with good system design in general, rather than design issues unique to AI. This is as it should be. As AI enters mainstream use, AI techniques become another set of methods in the repository of software solutions. The flip side of this is that AI systems are not immune to the problems associated with design for complex applications. Detailed problem analysis is the only way to find a good match between these techniques and the applications. The specifics of the problem, and the operational context into which the system is embedded must help to determine the form of the solution proposed.

References

Berner, C.A. Durham, R. & Reilly, N.B. (1989) Ground data systems resource allocation process. *Goddard Conference on Space Applications of Artificial Intelligence.* Greenbelt, MD. NASA Conference Publication 3033, 37-47.

Britt, D.L. Geoffroy, A.L. & Gohring, J.R. (1990) Managing temporal relations. *Goddard Conference on Space Applications of Artificial Intelligence (these proceedings).*

Graham, R.L. (March, 1978) The combinatorial mathematics of scheduling. *Scientific American.* 124-132.

Ibaraki T. & Katoh, N. (1988) *Resource allocation problems: algorithmic approaches.* Cambridge, MA: MIT Press.

Parunak, H. (1987) Why scheduling is hard (and how to do it anyway). *Proceedings of the 1987 Material Handling Focus.* Georgia Institute of Technology.

Smith, S.F. Fox, M.S. & Ow, P.S. (Fall, 1986) Constructing and maintaining detailed production plans: investigations into the development of knowledge-based factory scheduling systems. *AI Magazine*, 45-61.

PLAN-IT-2 The Next Generation Planning & Scheduling Tool

by

William C. Eggemeyer
Jennifer W. Cruz

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

Plan-IT is a scheduling program which has been demonstrated and evaluated in a variety of scheduling domains (Spacelab, Deep Space Network, Comet Rendezvous and Asteroid Flyby, Telescience, Space Station Power). This paper discusses the capability enhancements being made for the next generation of Plan-IT, called Plan-IT-2. Plan-IT-2 represents a complete rewrite of the original Plan-IT incorporating major changes as suggested by our application experiences with the original Plan-IT. A few of the enhancements described in the paper are additional types of constraints, such as states and resettable-depletables (batteries), dependencies between constraints, multiple levels of activity planning during the scheduling process, pattern constraint searching for opportunities as opposed to just minimizing the amount of conflicts, additional customization construction features for display and handling of diverse multiple time systems, and reduction in both the size and the complexity for creating the knowledgebase to address the different problem domains.

As the complexity of software on future spacecraft increases, so will planning and scheduling activities for the spacecraft. For a ground support tool to handle the scheduling/planning for spacecraft from manual to automatic operation the tool must: 1) contain multiple levels of planning from goal planning down to the detailed command level; 2) be adaptable to changes in the structure of the activities; 3) consider the resource constraints at multiple planning levels; 4) be adaptable to changing rules and strategies as mission goals and requirements change; 5) be user-natural (program operation is both user-friendly and intuitive to the user). Plan-IT-2 is the second step in bringing this capability to the sequencing domain. It is expected that experience gained with Plan-IT-2 will serve as good preparation for coping with spacecraft systems with various levels of autonomous behavior.

INTRODUCTION

Plan Integrated Timelines (Plan-IT) [7] is a scheduling tool coded in LISP that has been used and demonstrated on various projects. The major success of the program has been its ability to enhance the human schedulers' ability not only to produce an acceptable schedule more quickly than before but also to make adjustments to the schedule dynamically. The cognitive approach of encoding user-visualizations of constraints and a timeline view of the schedule makes this possible. Knowledge gained from the past scheduling experiences has led to the development of Plan-IT-2. This paper reviews Plan-IT experiences, and then discusses what has been learned from those experiences. The main portion of the paper describes Plan-IT-2's capabilities and representations.

PLAN-IT

Plan-IT was our first attempt to cognitively address activity scheduling for spacecraft missions. The approach of the program was to mimic how a human scheduler visualizes the problem to simplify resolving conflicts that occur during the scheduling process. This visualization took the form of a timeline oriented display for both the activities to be scheduled and the constraints considered in the schedule. Plan-IT also differed from previous approaches in that the representation allowed conflicts to exist in the schedule. The tool supported a range of scheduling capabilities from interactive manual editing up to automatic scheduling by strategy invocation.

Plan-IT Background

The Sequence Automation Research Group (SARG) originally had demonstrated the potential use of Artificial Intelligence (AI) concepts in the field of sequencing during Voyager Uranus encounter planning by using a program called DEVISER [2] [3] to create Voyager sequences at an intermediate level of detail. This program took a backward-chaining goal expansion approach with a rule-type knowledge base to create a temporally instantiated tree of activities to achieve those goals. The success of the DEVISER demonstration sparked interest from management at Marshal Space Flight Center concerned with scheduling SpaceLab missions for the Shuttle. Unfortunately, the SpaceLab scheduling problem exploited the weaknesses in DEVISER's approach to planning while not taking advantage of the strengths. The original Plan-IT was developed in less than a year to overcome this problem [5] [6]. Because of Plan-IT's cognitive approach to the scheduling problem [7], Plan-IT was adapted to a number of projects [12] besides SpaceLab to demonstrate the usefulness of the Plan-IT concept. Below is brief description of those efforts.

SpaceLab

SpaceLab was a multi-year effort requiring extensive code additions to adapt Plan-IT to operate within an already existing scheduling system, called Experiment Scheduling Program (ESP). Plan-IT's task for this application was to permit the user to tweak an already existing schedule either graphically by manual tweaks or by algorithmic strategies specially coded for the SpaceLab problem domain. The SpaceLab experience demonstrated that the Plan-IT approach could handle their scheduling problem with a few caveats. The drawbacks were that adapting Plan-IT to SpaceLab's particular problem domain took an excessive amount of time and required extensive coding modifications to Plan-IT's internal representations. These drawbacks indicated that Plan-IT's internal structures were not robust enough for easy adaptation to different problem domains.

Space Station Power Scheduling Proof of Concept

The Space Station Power Scheduling demonstration [4] was one of the first successes of Plan-IT's approach. This application required Plan-IT to work with simple prioritized activities and real-time dynamic changes to update the schedule as changes occurred during its execution. Adaptation to the problem domain was easy because of its restrictive nature, but it also indicated the need for additional types of resource constraints that Plan-IT did not model.

Deep Space Network (DSN) Application

Plan-IT was adapted in six months for scheduling the allocation of DSN radio-dishes around the world [8]. Plan-IT enhancements developed for this problem domain included easing the user edits, handling of generic as well as specific requests, and specialized algorithmic strategies. Plan-IT was used as an interim solution to the DSN scheduling problem until the Resource Allocation Planning Helper (RALPH) system development was completed [13]. Plan-IT successfully demonstrated its capabilities by reducing the DSN turn around time for scheduling by an order of magnitude. In spite of the improvement in turnaround time for scheduling, it was found that Plan-IT's global algorithmic strategies (unlike the specially coded strategies that were local in scope) either did too much or too little to the schedule. This illustrated that Plan-IT's global strategy algorithms are too brute force in nature because of their programmatic approach.

Comet Rendezvous Asteroid Flyby (CRAF) Demonstration

This was the first successful demonstration of Plan-IT's application to deep space missions [9]. Additionally, Plan-IT was combined with a natural language understanding system [10] [11], enabling Plan-IT to take requests for the spacecraft in English form and translate them into activities which then can be scheduled. This demonstrated to JPL management that such a "user-natural" scheduling system can have significant contributions to the spacecraft command and control process. The Plan-IT concepts proven by the demonstration are presently being implemented in a C-based version, called SFOC Planner, on the Sun microsystem workstation for JPL's Space Flight Operations Center. Early estimates showed a projected cost savings of $4 Million for the planned CRAF and Cassini missions due to the SFOC Planner implementation.

Telescience Demonstration

This demonstration was a joint JPL and Goddard Space Flight Center (GSFC) effort. GSFC provided the user-interface into the scheduling system, and JPL provided PLAN-IT, the scheduling tool. This task demonstrated that additional research into the field of peer teleconferencing is greatly needed. Allowing multiple users to concurrently interact from their home institutions on a schedule of activities raises a few issues: 1) how are multiple user simultaneous and cooperative edits to the schedule to be handled; 2) how is control of the scheduling session handled; 3) how should security be handled to guarantee privacy among the users, and; 4) how should the database be maintained. These and other Telescience issues will be addressed in a proposed future effort.

Knowledge Gained from Past Experiences

Scheduling in various problem domains has led us to the following conclusions about capabilities required for a scheduling tool. Some of these requirements may seem trivial and obvious, but their inclusion may make or break the tool's acceptance. Plan-IT-2 is being developed with extensive changes to both the internal representations and conceptual operation of the program to address the issues below.

Visualization and Adaptation Issues

The activities, resource constraints, and display layout must be dynamically configurable. For the various problem domains that Plan-IT was applied to, the display had to change in one form or another to meet the desired requirements of the users. Even within the same problem domain different users wish to view the schedule differently to enhance their visualization of the problem.

Time representation and its display must be changeable. Experiences with Plan-IT have shown that users like to see and work with multiple time systems. Some of these time systems can be very strange (ie: Galileo Command Data System time works by major and minor frames and realtime interrupts which are 60+2/3 second, 2/3 second and 2/30 second respectively), so a mechanism must be incorporated into the tool to allow the users not only to add unique time systems but also to work with and display multiple time systems.

Adaptation of the tool to the problem domains should be done with as high a level of language as possible. Plan-IT's adaptation process was overly complex and tedious. Each particular problem domain required the person adapting Plan-IT not only to be knowledgeable about Plan-IT but also to know how to program in LISP to overcome the restrictive internal representations present in Plan-IT.

Constraints and Representation Issues

There are many different types of resource constraints that a scheduling tool must handle. Plan-IT's repertoire of scheduling constraint types was limited and not complete. Additionally, some of the problem domains require complicated models, consisting of combinations of these simpler constraint types working in unison via dependency relationships.

In many of the problem domains there were requirements for handling temporally complex activities. Plan-IT's activity representation by only frames and slots [1] was not flexible enough for adequately representing these complex types of activities. An example of a complex activity is a generic request or a cyclic where an activity is supposed to occur multiple times every so often in time. Tweaking components of such a structure may have repercussions on the other components depending on the temporal dependencies involved. Defining complex activities in Plan-IT was a complicated and tedious programming task, leading to the need for a simpler and more robust method.

Scheduling Issues

Scheduling by algorithmic means works fine for simple restricted problem domains, but spacecraft activity scheduling requires a more robust approach. Users found that Plan-IT's strategies do either too much or not enough to the schedule even if the user constrained them through windowing and resource constraint consideration techniques. The users also found that traceability of the strategy execution was not intuitive enough to understand why particular actions occurred. Additionally, the task of creating new strategies required too much coding in LISP.

Users have multiple focus levels on the detail of the activities, along with their resource usage, in the schedule. Plan-IT scheduled activities at only one level of detail. Typically, as a schedule develops in an incremental fashion, users tend to work from a broad view of the activities and their resources to generate a preliminary schedule down through more levels of detail as the schedule is refined. Sometimes during this process the user's focus level may change back up to a more abstract level to resolve conflicts that arise. Plan-IT's single level focus operation was clearly a limitation to users who found it contextually limited for editing and for perceiving the problems within the schedule.

Users always want faster turnaround time in generating the schedule. Plan-IT's advantage in producing schedules faster was that users were able to concentrate on solving conflicts rather than taking time to identify the conflicts. Even though Plan-IT was much faster than DEVISER (scheduling speed decrease relative to the number of activities was linear vs. exponential), speed improvements can still be made.

PLAN-IT-2

The remaining sections of the paper concentrate on Plan-IT-2. The first is an overview of how the program interacts with the user through its five different independent processes. Following the process overview is a brief description of the file types the program supports, along with a very brief description of other user interaction capabilities with the program. The remaining portion of the paper concentrates on the actual Plan-IT-2 objects and their representation. Further elaboration of the definitions used for describing Plan-IT-2's objects can be found in [15].

Processes in Plan-IT-2

Plan-IT-2 operation is controlled by five different types of independent processes. These processes are created and invoked by the user, and monitor user-interaction with the program. The five processes are called the display, mouse-buffer, task-buffer, tactical process and activity process. Figure 1 describes the processes in Plan-IT-2.

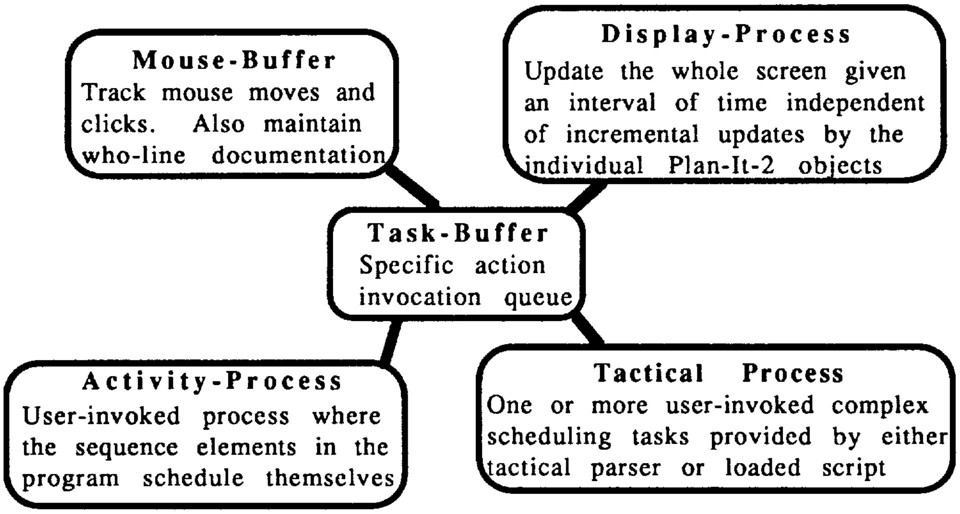

Mouse-Buffer
Track mouse moves and clicks. Also maintain who-line documentation
Display-Process
Update the whole screen given an interval of time independent of incremental updates by the individual Plan-It-2 objects
Task-Buffer
Specific action invocation queue
Activity-Process
User-invoked process where the sequence elements in the program schedule themselves
Tactical Process
One or more user-invoked complex scheduling tasks provided by either tactical parser or loaded script

Figure. 1 Plan-IT-2's Processes

Process Interaction

In Plan-IT-2 there are two processes called mouse and task buffer that always run in the background of the program. All actions invoked by the user and Plan-IT-2, varying from changing the scale of the display to invoking a tactical process, are executed from the task buffer. The other processes may be user-invoked at any time during the scheduling session.

Scheduling Processes

The user can invoke automatic scheduling in three ways. Two of these scheduling ways are performed by the tactical process, while the third way is handled by the activity process.

The tactical process is generated by the user in two different ways. The first way is by invoking a tactical natural language parser that accepts from the user a sentence in a simple language describing the algorithm of scheduling he wants to perform. A simple sentence example that would duplicate the old Plan-IT shuffle strategy is, "For all classes of activities move while conflict". A little more complicated example would be, "For the classes meta-activity and activity move, spawn and slink considering power and camera while conflict". This would cause Plan-IT-2 to determine the subset of all instances of the type meta-activity and activity if they are involved with a conflict for the power and camera resource constraints. Then for each individual activity or meta-activity instance within that subset of activities, move to the most receptive place in the schedule. If conflict is still present for the individual activity then focus down a level in detail to its sub-components, and then if there is still conflict try to slink (flex its structure) to eliminate the conflict. If the user worded the tactical command in the form of "For classes meta-activity and activity move then spawn then slink considering power and camera while conflict" then the program would form a subset of the activities for actions as before, but the order invoking those actions would change. The program would loop through the subset of activities and meta-activities three times instead of once applying the requested action if the activity instances were in conflict. So, instead of moving, spawning and slinking on each activity instance before going to the next instance in the set, Plan-IT-2 applies each action to the whole set of activity instances before going to the next action. This demonstrates the relative ease for a user to generate scheduling algorithms at his own level of understanding, rather than by unintelligible predefined hard-coded routines.

The second way of invoking the tactical process is to read a file that scripts out the tactical commands to perform in order. The user is permitted to execute multiple tactical processes simultaneously; however, interaction between them may cause trouble. Plan-IT-2 is being coded so that it will also be a useful testbed for testing how different aspects of its operations would work in parallel.

The last way that the Plan-IT-2 can schedule is the activity process. This process allows the activities from their own perspective to try to fix the conflicts in the schedule. This will be explained in further detail under the Activity Representation section.

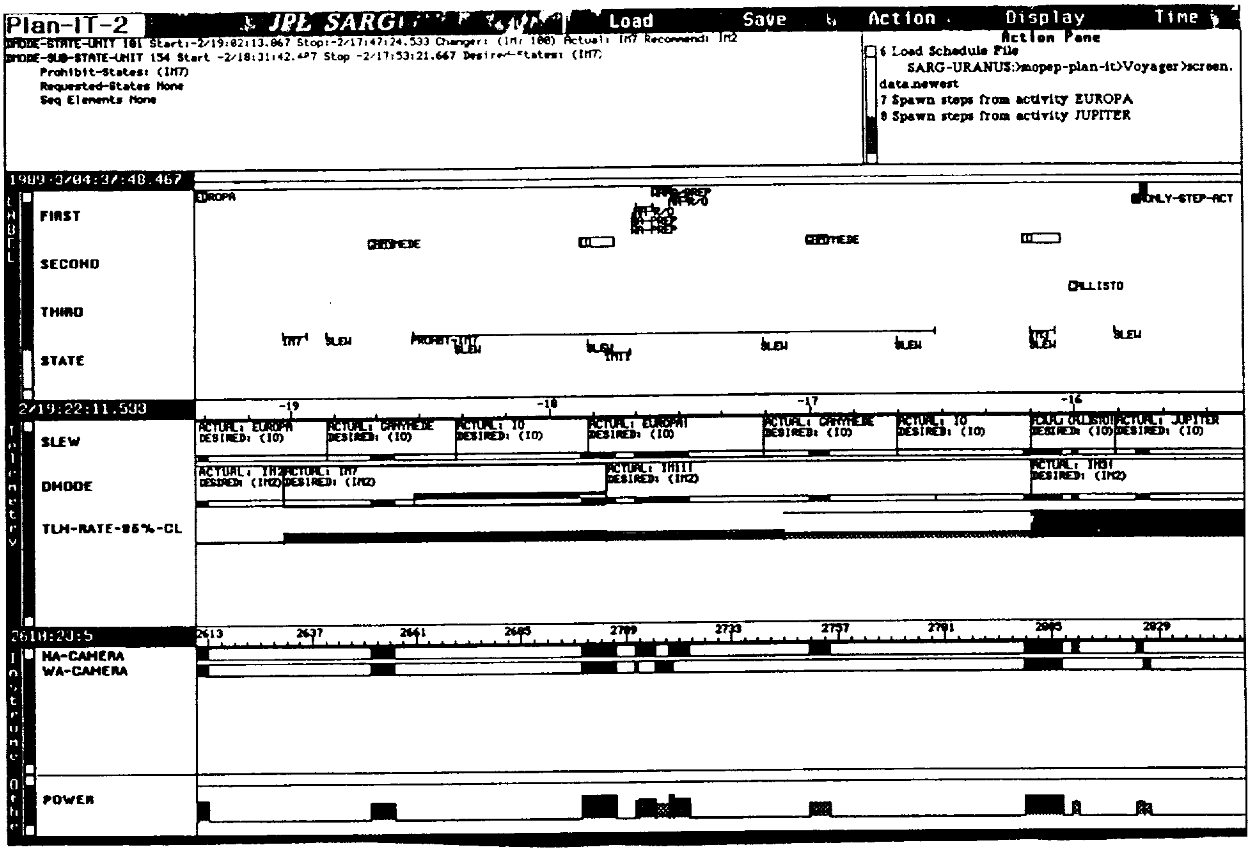
File Types

File I/O capabilities of the program have been increased to accept the following types of files: 1) display - establishes the display setup (activity displaying panes, resource constraint displaying panes, etc.), time resolution, the start time of the schedule, and the time systems to be use; 2) legend - manipulates where activities are to be vertically located within each activity displaying pane (more than one is now allowed); 3) setup - defines the resource constraints which are to be used, which constraint pane they belong to, optional initialization parameters, and the duration of the schedule; 4) script - contains a batch execution file of commands and actions to execute in the program; 5) data - activity data that the program schedules; 6) project - load problem definition system and optionally; 7) owlt - contains one-way light time data for conversion from ground to spacecraft time. All of these file types, with the exception of the project definition type, are for both input and output and are in human readable form.

Other Interaction Capabilities

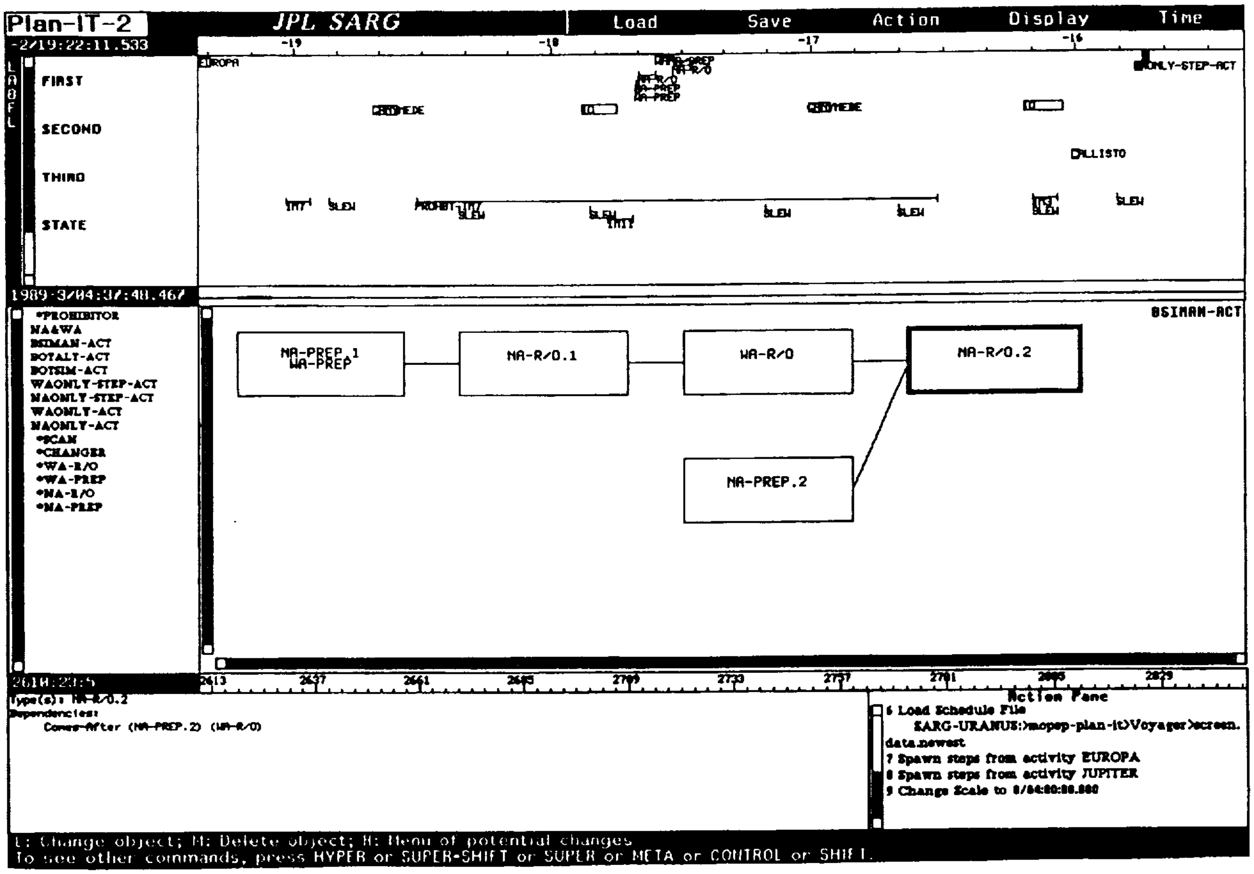
The manipulation of Plan-IT-2's display is built into the tool itself. The user can access a graphical display editor to modify the display format in realtime, to load a display file for possible editing, or to save a newly generated display format. In addition to giving the user the flexibility of modifying the display, the user may also graphically create, modify, and save activity types for the schedule. This is accomplished by another built in graphical editor (explained in the Activity Representation section). Two forms of Plan-IT-2's display are illustrated on the following page.

Plan-IT-2 also contains an action pane which keeps a verbatim history of what actions both the user and the program perform on the schedule. The contents within the action pane may be selectively saved to a script type file for other scheduling sessions, or may be scanned through to re-execute a specific command or action by a mouse click.



```
DMODE-STATE-UNIT 181 Start:-2/19:02:13.867 Stop:-2/17:47:24.533 Changer: (IM: 100) Actual: IM7 Recommend: IM2
DMODE-SUB-STATE-UNIT 154 Start -2/18:31:42.497 Stop -2/17:53:21.667 Desired-States: (IM7)
     Prohibit-States: (IM7)
     Requested-States None
     Seq Elements None
```

```
Action Pane
6 Load Schedule File
   SARG-URANUS:>mopep-plan-it>Voyager>screen.
data.newest
7 Spawn steps from activity EUROPA
8 Spawn steps from activity JUPITER
```



```
Type(s): NA-R/0.2
Dependencies:
     Comes-After (NA-PREP.2) (WA-R/O)
```

```
Action Pane
6 Load Schedule File
   SARG-URANUS:>mopep-plan-it>Voyager>screen.
data.newest
7 Spawn steps from activity EUROPA
8 Spawn steps from activity JUPITER
9 Change Scale to 0/04:00:00.000
```

```
L: Change object; M: Delete object; R: Menu of potential changes
To see other commands, press HYPER or SUPER-SHIFT or SUPER or META or CONTROL or SHIFT.
```

Plan-IT-2 Normal Display (top) and Graph Editor Display (bottom)

ORIGINAL PAGE IS
OF POOR QUALITY

Resource Constraint Capabilities

The planning/scheduling problems of spacecraft require a multitude of different types of resource constraints. Plan-IT-2's resource constraint representation is more complete than that of the original Plan-IT. The resource constraints exist as timelines on Plan-IT-2's display illustrating exactly how they are represented internally to the program. The main job performed by all of these timelines is to maintain a breakdown of the unique list of temporally intersecting activities in the schedule as illustrated in the figure 2.

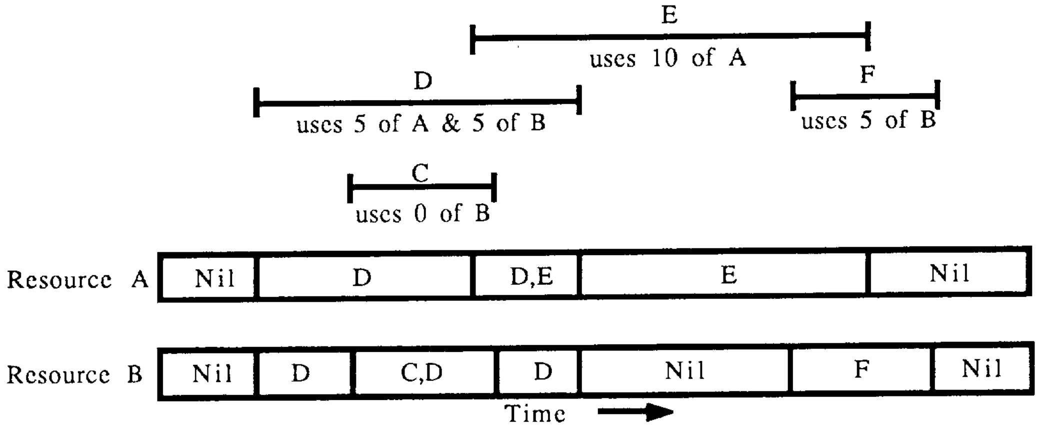


Figure. 2 Activities C, D, E and F monitored by resource constraints A and B

Constraint Dependency Mechanism

All resource constraint types have the ability to influence other resource constraints through two types of dependency mechanisms that are concerned with the constraints' usage state. The usage state is typically a histogram breakdown of the usage of the resource constraint over time. But because the usage state may not change as the unique list of intersecting activities in time changes for that resource constraint, it may not necessarily have a one-to-one correspondence with how the resource constraint line itself is divided up. When a dependency exists between the resource constraints, the resource constraints generate and maintain dependency events or daemons between themselves. These dependency events exist in two basic forms. The simplest is a uni-directional dependency in which one resource constraint directly influences another by its usage state. A simple example of this would be a power and energy constraint system. The power and energy system would consist of two simple resource constraints, power and energy. There would exist a uni-directional dependency going from the power resource to the energy resource in the form of multiplying the unique power amounts by their respective durations of existence to determine the energy consumed. The power constraint looks for exceeding the threshold of available power as the energy constraint checks if the threshold amount of energy available for the schedule is over-utilized. Figure 3 illustrates the dependency mechanism given a uni-directional dependency between the resource constraint B in the previous figure and another constraint called C.

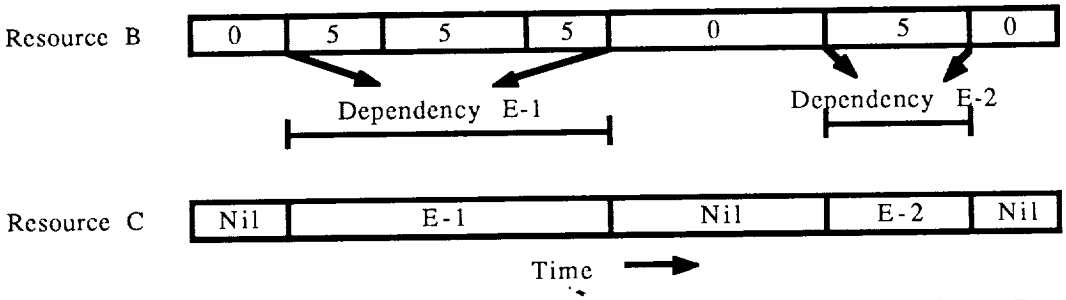


Figure. 3 Uni-directional generated dependency events monitored from B to C

The more complicated dependency is called a bi-directional dependency. Its job is to link two resource constraints together, as the uni-directional does, but given the condition of both timelines it determines which one to influence and by how much. Both dependency mechanisms create the same type of primitive dependency event that is monitored by the resource constraints. This daemon is monitored the same way as are the activities within the schedule.

Concurrencies

The simplest resource constraint in Plan-IT-2 is called concurrency. This type of resource constraint exists in two forms, called *concurrency* and *non-concurrency*. Both types of timelines monitor two types of activities in the schedule. One type of activity indicates the presence of something while another type of activity indicates the need for it. The concurrency constraint looks for matching the two types of activities together. If an activity of one type needs whatever the constraint represents at a time where that constraint is not present then there is a conflict. The non-concurrency constraint operates in just the opposite manner. Non-concurrency wants no intersection in time between those types of activities that need the constraint with those that indicate its presence.

Non-Depletables

Non-Depletables are non-consumable types of resource constraints that automatically restore themselves when they are not in use. This resource constraint was pioneered by the original Plan-IT program. Non-depletables exist in Plan-IT-2 in several forms. The simplest one called *availability* monitors a resource constraint such as whether a camera is used more than once simultaneously. The next most complicated type called *non-depletable-step* is concerned with an amount of something such as power being oversubscribed at any one time (note: that this limiting amount may itself vary over time but in a step-wise fashion). Finally there is the continuous one called *non-depletable-continuous* which is similar to non-depletable-step except the amount's limit is determined by some continuous function (eg: the shuttle's thermal constraint).

Depletables

Depletables are a consumable type of resource constraint that DOES NOT restore itself when not in use. As with non-depletable types there are several forms of depletables. The simplest called *depletable* starts with a fixed amount of something and through a step-wise fixed consumption the amount gradually becomes depleted. A simple example of this is modelling fuel usage on a spacecraft in a step-wise fashion. The spacecraft starts out with a fixed amount and as it is being consumed by the thrusters during maneuvers it gradually becomes depleted.

The next most complicated one called *resettable-depletable* operates just as a depletable does, except that it permits certain activities or other resource constraint dependencies to replenish the amount. This replenishment may either be a partial or complete amount. A good example of a resettable-depletable type of constraint is a rechargeable battery.

The last and most complicated form of a depletable is the *positional-depletable.* This constraint acts similarly to the resettable-depletable by being both depletable and replenishable except that it is also concerned with the location of usage. An excellent example of this is a digital multi-track tape recorder (DTR) that allows positioning for recording and playing back data.

States

States are the most complex type of constraint. A state constraint is a mode or a condition of being. A simple example of this is a toggle switch that can either be in an on or off state. There are three state operators, called changer (an activity that changes the state), user (an activity influenced by a state), and prohibitor (an activity that prohibits certain states). Plan-IT-2's representation for a single state constraint actually consists of two interdependent timelines. One timeline keeps track of the time the state changes, the current state, state users concerned about overlapping the time of a state change, state changers occurring at the same time, the most desired state derived from all of the users within that state's duration and finally, times when the selected state is prohibited. The other timeline keeps track of the unique time intersecting activities in the schedule, a running sum of all of the potentially desired states by the users of this resource, lists of those activities that cannot use that selected state, and lists of activities whose desired states conflict with any prohibited states.

Activity Representation

Plan-IT-2's representation of activities in the schedule has dramatically changed from the original Plan-IT. Plan-IT-2's enhancements to the frame and slot structures elevated the user's understanding and maintenance of the activities in the schedule. As in the original Plan-IT, frames and slots are still used to hold the information that represents a single activity and all of its resource constraints. Additionally, the activity structure contains knowledge for scheduling itself. The activity attributes described below detail out these improvements.

Time Slot

Time representation has been separated from the rest of the slots to give the user a greater amount of flexibility for influencing the activities' choice of actions. The time slots in Plan-IT-2 permit the user to define temporal flexibility of the activity instances in the schedule. The time slots vary from just having a start, stop and duration value to having a flexible duration with multiple time windows containing multiple preference choices. Another new Plan-IT-2 feature is the ability to tag the time system type to the data during input so that it can be saved in the same time format.

Generic Slots

As before in the original Plan-IT, the slots in all of the activity types, represent the resource constraint utilization applicable to that activity. In the original Plan-IT the slots unfortunately required extensive coding to adequately represent the constraint usage for the activities. However the slots in Plan-IT-2 are now generic types requiring only a single form to define the slots' linkage to either a particular constraint or list of resource constraints applicable to the activity type. Table 1 gives the generic slot types.

Table 1. Thirteen different Plan-IT-2 Slots

Slot Type	Defines	Input Data
Single-Availability	What is needed or present	:Present or :Needed
Multiple-Availability	List of what is needed or present	List and :Present or :Needed
Amount	Usage	Number or function
Varying-Amount	Range of usage	Number range or function and choice
Reset-Amount	Amount to replenish	Number or symbol or function
Reset-Varying-Amount	Range amount to replenish	Number range or function and choice
Simple	Resource in use	N/A
Multiple-Choice	A list of resources to use	List or function
State-Changer	Changing resource to a state	Change state and list or function
State-User	Desired state from state resource	Desired state and list or function
State-Prohibitor	States to avoid for state resource	List of states or function
Priority	Priority	Number or symbol or function
Info	Other information on the activity	Text

Each slot is capable of reading in, being edited, and writing out its contents within the context of the activity using it. Some of the slots change themselves appropriately depending on the scheduling actions applied to them. Presently, the functions invoked by slots are passed the instance to execute on and a time value. Additional slot options are the initialization parameters for the slot type, an ordering precedence for how the frame structure displays a slot relative to other slots, and the slot utilization by the activity type definition.

Example 1 illustrates definitions of multiple choice, simple and state slots for the narrow angle (NA) and wide angle (WA) camera system and for some databus telemetry modes (DMODE) used by the Voyager spacecraft.

```
(Define-Slot-Type Instrument Multiple-Choice 2 t  :list-of-choices  '(na-camera wa-camera))
(Define-Slot-Type Na-Camera Simple 2 t)
(Define-Slot-Type Wa-Camera Simple 2 t)
(Define-Slot-Type Dmode-Changer State-Changer 2  nil  :name DMODE  :state-list  '(gs3 im2 im7 im11 oc1))
(Define-Slot-Type Dmode-User State-User 2  t  name  DMODE)
(Define-Slot-Type Dmode-Prohibitor State-Prohibitor 2  nil  :name  DMODE)
```

Example 1. Some Slots for the Voyager Problem Domain

In example 1, the slots fit into two basic categories called shared and non-shared slots. The t or nil following the ordering precedence number indicates whether or not that slot is to be shared by all of the components of the activity structure that uses that slot. For both speed and memory considerations Plan-IT-2 allows the user to control how the activity structures are constructed with the slots. The shared attribute even has a global option for sharing, so for a given problem domain all activity instances using that slot will be using the same slot instance. Depending on how the slot is defined by the user, there is a wide variety in the scope of any changes made to that slot during the scheduling process. For instance, if the slot is defined as being globally shared, a change to that slot implies a change to every activity instance using that slot.

Activity Types

Activities in Plan-IT-2 have been redefined into five specific types of objects called event, step, activity-step, activity and meta-activity in ascending order of complexity. All of these object types contain a time slot whose type influences the activity's execution of different scheduling actions. Slots representing the usage of particular resources may optionally be included in any of these five activity type structures.

An event is the simplest and easiest type of activity to schedule. The event represents a single level of detail for the resource usage for its duration and is not dependent on anything else in the schedule with the exception of time.

A step is exactly like an event, except that it has a more abstract parent object controlling it. The step represents the most detailed level of resource usage for either an activity-step, activity, or meta-activity object. Any temporal dependencies involved with that step are controlled by its parent.

An activity-step provides an intermediate level of abstraction between a step and either an activity or a meta-activity. The activity-step can contain slots representing resource usages at that level and may have other activity-steps as either its parent or children. There is essentially no limit to the number of intermediate levels of activity-steps a user may define in Plan-IT-2.

Both activity and meta-activity are the most abstract objects that Plan-IT-2 schedules. They may optionally contain slots for resource usages to be considered at their level of abstraction for the overall activity structure that they reside on top of. The major difference between activity and meta-activity is the way they control the monitoring of their slots by the resource constraint timelines.

The user can define his own activity type for Plan-IT-2 built upon these basic types with a simple form. This form specifies the activity type, its time capabilities, display attribute options (except for meta-activity), slot type attributes, component relationships (except for event and step), and default values for its slots. Additionally, the timing requirements and action capabilities (such as move, shrink, etc.) are assigned to the five activity types. These simple forms are illustrated in the examples following the next section.

Activity Node Network Structure

Each activity type, except those created from the event type, is represented by a specialized node network. Each node within this network contains information representing both temporal and functional relationships between a node and its neighboring nodes. There are twelve fields per node. One field of a node maintains a list of one or more activity types that are represented by the node in the network. These activity types may be at intermediate levels of abstraction in which they themselves consist of their own node networks. Another field represents how the activities could be repeated (every so often, during something, how many times, etc.). Seven other fields hold pointers to other nodes each representing a specialized form of temporal relationship with this node. These relationships are: 1) comes-before; 2) starts-before; 3) comes-after; 4) starts-with; 5) ends-with; 6) during; 7) not-during. Another field represents the concept of OR in the network. This field gives the network the ability to handle activities that may be multi-configurable in their structure. The graphical representation of these relationships used by the network graph editor in Plan-IT-2 is illustrated in figure 4.

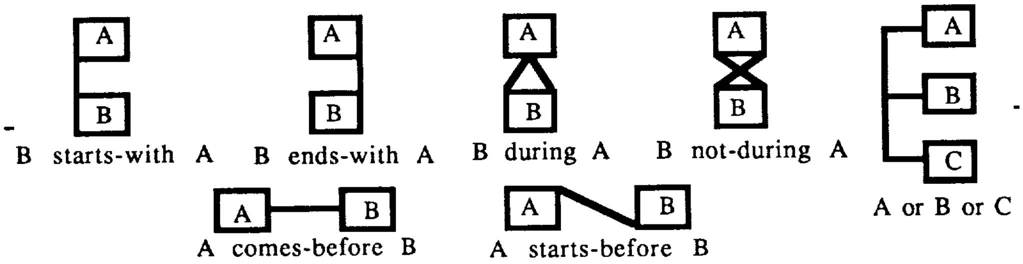


Figure. 4 The seven nodal relationships for an activity network

The two remaining fields of a node are called BY and IF. The temporal relationship specifics of these seven fields is controlled by the BY field. This field consists of an association list containing the nodes from the other seven fields and their specific temporal relationships with this node. The final field of the node is the IF field. This is similar to the BY field except that instead of holding temporal relationship information associated with the neighboring nodes it contains specific conditions to determine the linking of the node with its neighboring nodes.

The activity node networks can be both created and modified by the user before, during (causing changes in the schedule) and after the scheduling process in the program. This can be done graphically through the network graph editor built into the program or by textually typing in the simple form definition.

Example of Defining Activity Types

To illustrate the clarity and ease of this approach, we define an imaging activity for simultaneously shuttering both Voyager cameras three times. The constraints considered for this activity type are the cameras and the databus telemetry mode state. Below is a breakdown of how the representations would look textually as well as graphically, using the same slots defined in example 1.

In example 2, the most detail level defines the shuttering and image data readout steps for both cameras. Note the durations of the readouts are determined by a LISP function that is concerned with the databus telemetry state. Since it is a shared slot the dmode-user will be defined at the top-most abstract level of the activity structure that uses it.

```
(Define-Step Na-Prep basic-time () (Na-Camera Dmode-User) ((Duration "00:48")))
(Define-Step Na-R/O Duration-Range () (Na-Camera Dmode-User)
            ((Duration-Range Determine-Duration-From-Data-Mode)))
(Define-Step Wa-Prep basic-time () (Wa-Camera Dmode-User) ((Duration "00:48")))
(Define-Step Wa-R/O Duration-Range () (Wa-Camera Dmode-User)
            ((Duration-Range Determine-Duration-From-Data-Mode)))
```

Example 2. Step Definitions for an Imaging Activity

In example 3, the intermediate level of detail is an activity object but the consideration of the camera constraints is for both of them over its duration. This activity type's duration is also functionally dependent.

```
(Define-Step-Activity Botsim-Activity Duration-Range () (instrument Dmode-User)
                      ("Na-Prep and Wa-Prep comes-before Na-R/O"
                       "Na-R/O comes-before Wa-R/O")
                      ((Duration-Range Determine-Botsim-Duration-From-Data-Mode)
                       (instrument 2 (Wa-Camera Na-Camera) (Wa-Camera Na-Camera))))
```

Example 3. Activity Definition for Intermediate Level of Detail

Finally in example 4, the top abstract level of the imaging activity, the camera constraints are not even considered, but the default values for the dmode-user are given. Note that dmode-user is used for the activity structure construction name but it is reference as dmode in the default template form because that is the actual name of the constraint it is concerned with.

```
(Define-Activity 3-Pairs-Of-Simultaneous-Shutterings Duration-Range (Dmode-User)
                 ("Botsim-Activity repeats 3 times every Determine-Botsim-Duration-From-Data-Mode")
                ((Dmode im2 im7 im11)))
```

Example 4. Most Abstract Activity Definition

Figure 5 illustrates the complete graphical representation of this activity structure.

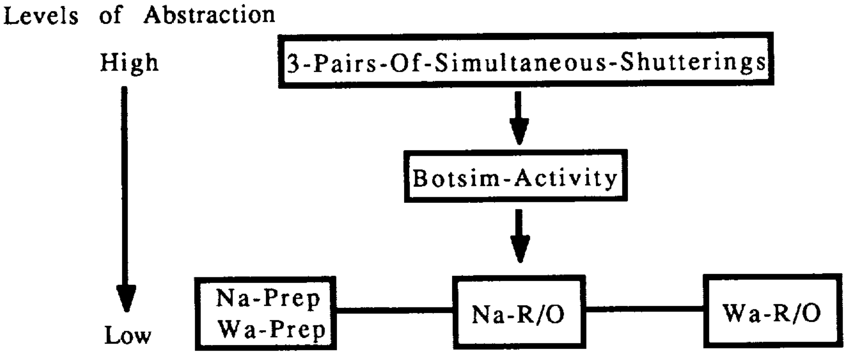


Figure 5. Abstract layering of 3-Pairs-Of-Simultaneous-Shutterings

There are some important aspects of the way the program handles this structure. First there is only one instance of the node network per activity type. When multiple instances of each activity type are instantiated, each creates a special list structure that instantiates a specific path through this node network with the actual children instances of the activity types specified within the nodes. This instantiation list also contains specific temporal flexibility information that exists between those children instances, so that the structure knows how to contextually control both Plan-IT-2 actions and user modifications to it.

This representation gives Plan-IT-2 two important capabilities that were lacking in the original Plan-IT. First is the ability to plan and schedule a sequence from any number of focus levels in much the same way a human scheduler works. Second, the robust activity definition capability eases adaptation of Plan-IT-2 to all of the known activity scheduling problem domains.

Generic Action Definition

The remaining information the user must define for an activity structure is the set of valid scheduling actions that are permitted. Plan-IT-2's new approach to scheduling has vastly changed from the old programmatic approach of the original Plan-IT. There exists within Plan-IT-2 a library of human-comprehensible scheduling actions (move, move-to, shrink, change-slot, change-self, distribute-self, reconfigure, slink, etc.) that may be invoked on the activities in the schedule.

There are two objectives for this approach: 1) scheduling actions are both traceable and executable in terminology palatable to the user rather than obtuse programmatic algorithms; 2) scheduling tasks are defined in more abstract terms leaving the local details to be handled by the objects themselves. For example, a generic action like *move* can be controlled contextually by the structure it is invoked on and also by the types of constraints and other dependencies it was told to consider.

User modifications on the activity structures are handled in much the same way. For example, if a user moves an intermediate-level abstract activity instance by the mouse. If this instance, being a child to a complicated activity structure, was moved beyond its allowable flexibility definition within the node network, or moved into another state that caused it to change, the whole layered structure of the node networks updates itself appropriately.

Remaining Plan-IT-2 Objects Used in Scheduling

Four remaining objects used by Plan-IT-2 for the scheduling process are mediators, scouts, short-term memories and rule monitoring. Mediators are objects that group conflicts over multiple constraints in a temporal fashion. The mediator's first job is classifying the conflict group into an abstract form. Once classification is complete, the mediator then determines which activities are involved with that conflict group. The mediator may optionally query the activities for information concerning their flexibility for taking action and what actions they are capable of. The mediator then uses the information available to suggest actions to activities for reducing or eliminating that conflict group. After several activities take the actions suggested by the mediators, the mediators will be regenerated.

Scouts perform resource usage pattern searches across the temporal areas of interest for the particular activity that originates them. It is the job of these scouts to receive openness reports for their temporal location from each of the constraints involved. Each resource reports this openness within its own predefined normalized form. The scout merges these reports together in a final report for the activity. This report is from an opportunistic perspective because of the way the resources responded to the scout's request. If the action derived by the scout's report results in escaping the conflicting situation for that activity without effecting dependent neighboring activities, the activity would immediately execute the action. However, if the action's effects do ripple beyond that activity, then the activity must report to its parent what it desires to do and wait for the parent to decide if the action should be done at that level, or at a higher abstract level, or done differently, or even done at all.

Both the short-term memory for the activity structures and the monitoring of rules have yet to be finalized in form. The main objective for the short-term memory is to influence the action decision process of both the activities and the mediators. Rule monitoring will be a user-invoked independent process that will apply defined heuristics to the schedule. Here is an example of a heuristic concerned with the activity structure related to resource utilization. If there are enough top level activities of a particular type consuming a depletable resource (such as memory bytes) and the use of that resource can be reduced by making the activity instances part of a meta-activity, then change their structure appropriately and create the parent meta-activity.

Plan-IT-2 Mode of Execution

Plan-IT-2 uses conflicts in the schedule only as motivators for taking scheduling actions. The conflicts themselves are no longer used to determine the success of a scheduling action. The monitoring of success of an action is left to both the activities themselves and the user monitoring the program. The execution of the scheduling actions relies on viewing across multiple constraint timelines based upon the opportunistic view presented by the constraints, merged together by the scouts for the activities. Unlike the original Plan-IT, there is no form of global measurement of goodness for the schedule by the constraints. Presently, Plan-IT-2 executes these actions serially from the task buffer. When Plan-IT-2 is completed an attempt to parallelize the automatic determination and execution of non-interfering actions will be made.

SUMMARY

Plan-IT-2 is our first system to address all of the issues involved with generic activity scheduling. From the early days of our DEVISER experience with Voyager, we learned that AI concepts were applicable to spacecraft sequencing. Experience gained by the application of the original Plan-IT in other activity scheduling domains further evolved our scheduling concepts to the structures and representations in Plan-IT-2. A comparison of the estimated amount of adaptation required for Plan-IT-2 when completed vs. DEVISER for Voyager class problems illustrates how astounding the advancements are. DEVISER required about 45 pages in rules and a few additional pages for domain specific LISP functions to address Voyager activity scheduling. Plan-IT-2 is estimated to handle the Voyager scheduling problem with about 10 pages for its knowledgebase and domain specific functions. This is due to the inherent robustness in Plan-IT-2 structures and representations. Plan-IT-2 is also attempting to address faster turnaround time for scheduling, both by code optimization and by the new approach. Finally, both the object-oriented design and conceptual operation of Plan-IT-2 makes it a good platform for research on addressing the scheduling problem with parallel architectures.

ACKNOWLEDGEMENTS

The research described in this paper was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

Other individuals participating in this project include R. Brill, C. Collins, W. Dias, J. George, S. Grenander, M. Hollander, W. Lombard, J. O'Meara, D. Mittman, S. Peters, M. Rokey, J. Sisino, E. Zamani, and B. Zimmerman.

REFERENCES

1) *The Handbook of Artificial Intelligence*, Barr, A. and Feigenbaum, 1981, Vol. 1 p. 156.

2) "Planning in Time: Windows and Durations for Activities and Goals", S. Vere, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5, 3, 1983.

3) "Toward the Fully Capable AI Space Mission Planner," S. U. Grenander, *Aerospace America*, August 1985.

4) "Space Power System Scheduling using and Expert System," K. Bahrami, E. Biefeld, L. Costello, J. Clein, Proceedings, *21st Intersociety Energy Conversion Engineering Conference*, San Diego, CA, August, 1986.

5) "Outward Bound: Machine Intelligence in Deep Space", S. Grenander, *Computers in Mechanical Engineering*, September, 1986.

6) "Artificial Intelligence Planning Applications for Space Exploration and Space Robotics", M. Rokey, S. Grenander, *Aerospace Applications of Artificial Intelligence*, Conference Proceedings, Dayton, Ohio, October 1986.

7) "PLAN-IT: Knowledge-Based Mission Sequencing," E. Biefeld, Proceedings, *Advances in Intelligent Robotics Systems Conference*, Cambridge, Mass., October, 1986.

8) "Deep Space Network Resource Scheduling Approach and Application," W. Eggemeyer, A. Bowling, Proceedings, *Space Applications of AI and Robotics*, GSFC, May, 1987.

9) "Plan-It: Scheduling Assistant for Solar System Exploration," W. Dias, J. Henricks, J. Wong, *Telematics and Informatics*, Vol. 4, No. 4, pp. 275-287, 1987.

10) "Understanding Natural Language for Spacecraft Sequencing," B. Katz, R. Brooks, *Spaceflight, the Journal of the British Interplanetary Society*, Nov., 1987.

11) "START Natural Language System", B. Katz, MIT AI Lab Memo, 1987.

12) "Plan-It Applications and Knowledge Gained", W. C. Eggemeyer, S. U. Grenander, *Workshop on Operations Planning and Scheduling Systems for the Space Station Era*, University of Colorado-Boulder, Boulder, Colorado, August, 1987.

13) "Ground Data System Resource Allocation Process", Carol Berner, Ralph Durham, Norman Reilly, *NASA Pub. 3033 for 1989 Goddard AI Conference*, May 16-17,1989.

14) "Spacecraft Activity Planning Tool Using Object Oriented Techniques", E. Zamani, J. George, C. Collins, B. Zimmerman, Proceedings, Tools '89, Paris, France Nov. 13-15, 1989.

15) *A Planning and Scheduling Lexicon*, Jennifer W. Cruz, William C. Eggemeyer, JPL Publication 89-25, Sept. 15, 1989.

An Approach To Rescheduling Activities Based On Determination of Priority and Disruptivity

Jeffrey L. Sponsler and Mark D. Johnston

Space Telescope Science Institute
3700 San Martin Drive, Baltimore, MD 21218 USA
(301) 338-4565 sponsler@stsci.edu

Abstract

A constraint-based scheduling system called SPIKE is being used to create long-term schedules for the Hubble Space Telescope. Feedback from the spacecraft or from other ground support systems may invalidate some scheduling decisions and those activities concerned must be reconsidered. A function *rescheduling priority* is defined which for a given activity performs a heuristic analysis and produces a relative numerical value which is used to rank all such entities in the order that they should be rescheduled. A function *disruptivity* is also defined that is used to place a relative numeric value on how much a pre-existing schedule would be changed in order to reschedule an activity. Using these functions, two algorithms (a *stochastic neural network approach* and an *exhaustive search approach*) are proposed to find the best place to reschedule an activity. Prototypes have been implemented and preliminary testing reveals that the the exhaustive technique produces only marginally better results at much greater computational cost.

1. Introduction

Scheduling is an intellectual activity that humans do on a daily basis. Often this activity is accomplished without the awareness that reasonable (but not necessarily optimal) solutions are formulated for a generally hard problem. One may argue that the number of activities to be scheduled in a day is small, the constraints to be imposed are simple, and thus the problem is tractable. On the other hand, it should be recalled that a fine-grained massively parallel architecture fine-tuned over epochs is at work. Rescheduling is perhaps an equally important activity due to the fact that schedules are rarely executed precisely as planned and therefore must be revised dynamically. It is the focus of this report.

1.1. Description of the HST

NASA's Hubble Space Telescope (HST) is an orbital observatory to be launched by the Space Shuttle *Discovery* in 1990. It has six scientific instruments and will provide greatly improved resolution and sensitivity because it will be above the earth's atmosphere. The Space Telescope Science Institute (STScI) is responsible for managing the ground-based scientific operations of HST. Proposals for observation of astronomical objects are submitted by astronomers (professional and amateur) and are processed by a series of software programs. An expert system called TRANSFORMATION processes proposal data and produces data structures organized by rules. An AI system called SPIKE is used to create long term schedules (for periods of one year or more). SPIKE feeds the data from one week of such a schedule to a system called SPSS that is used to create a finely detailed schedule. From this is derived specific spacecraft commands. For more details about HST, see Hall 1982.

1.2. SPSS and TRANSFORMATION

SPSS (Science Planning and Scheduling System) is the short-term scheduling software for HST. SPSS operates on entities known as *exposures*, *alignments*, *observation sets*, and *scheduling units* (SUs). The term *exposure* is defined simply as an observation of some object by a science instrument (SI). An *alignment* is a specification for pointing the spacecraft. Generally this pointing may start at one point and end at another but in practice usually is a single point. One or more exposures may be assigned to the same alignment. An *observation set* is composed of alignments where all exposures have the same *guide stars* (reference stars used to maintain exact pointing of HST). A *scheduling unit* (SU) is composed of one or more *observation sets* that conform to certain requirements (e.g., there is a *sequential nogap* specification between bordering exposures of different observation sets).

The TRANSFORMATION system has been developed at STScI to generate this data organization. It uses heuristics obtained by *operations astronomers* who have in the past manually generated the SPSS data structures. The input is a proposal file prepared and submitted remotely by an astronomer. Its output is used to populate the SPSS database as well as to provide the SPIKE system with the data needed to generate its schedules.

1.3. The SPIKE scheduling system

The SPIKE scheduling environment consists of the core SPIKE constraint based system, a user interface, and a neural network based algorithm used to search for optimal solutions to the ST long term scheduling problem (Miller, 1989). Descriptions of these subsystems follow.

1.3.1. The Constraint Based Scheduling System

The SPIKE system has been created as a general purpose scheduler and so specific references to other systems and even to spacecraft are abstracted. The system operates on *activities*, *constraints*, and *scheduling clusters* (groups of activities). The mapping of terms according to the pattern *SPSS term/SPIKE term* includes: exposure/activity, constraint/constraint, scheduling unit/scheduling cluster (often the term cluster is used). It is the case that within the body of this paper these analogous terms will be used interchangeably.

SPIKE processes information from TRANSFORMATION about targets (e.g., "crab nebula"), exposures (e.g., "crab nebula using planetary camera"), constraints (e.g, "A before B"), and the proposal data organization that SPSS requires.

The *suitability function* is a means for representing scheduling constraints and preferences (Johnston, 1990). The approach is numeric and provides a powerful way to represent the concept of "goodness over time." The SPIKE approach is extensible and the constraint knowledge is represented explicitly as objects (Flavors instances) with associated methods.

In the SPIKE domain scheduling is treated as a **constraint satisfaction problem**. Constraints may be either *absolute time constraints* ("execute the exposure only if the sun is not in the target path"), *relative time constraints* ("execute exposure A before exposure B"), or *resource constraints*. Such problems are known to be NP-complete (Garey, 1979) and so the exhaustive traversal of the entire search space is not computationally tractable if the number of scheduling clusters is large.

The term *dependency cluster* is defined as follows: Let S be a set of activities that are in a dependency cluster. An activity A is a member of S if one can traverse relative time constraint links to all other activities in S. Informally, the dependency cluster contains activities that directly or indirectly affect (via relative constraints) the other activities in the cluster.

Using the SPIKE scheduling tools, one may make a scheduling decision (i.e., a *commitment*) that restricts the times when a scheduling cluster may be scheduled. The scheduling system will propagate changes based on the relative constraints to other clusters that contain activities so linked. In general, the suitabilities of other activities within a dependency cluster will shrink reflecting the notion that available scheduling windows are smaller.

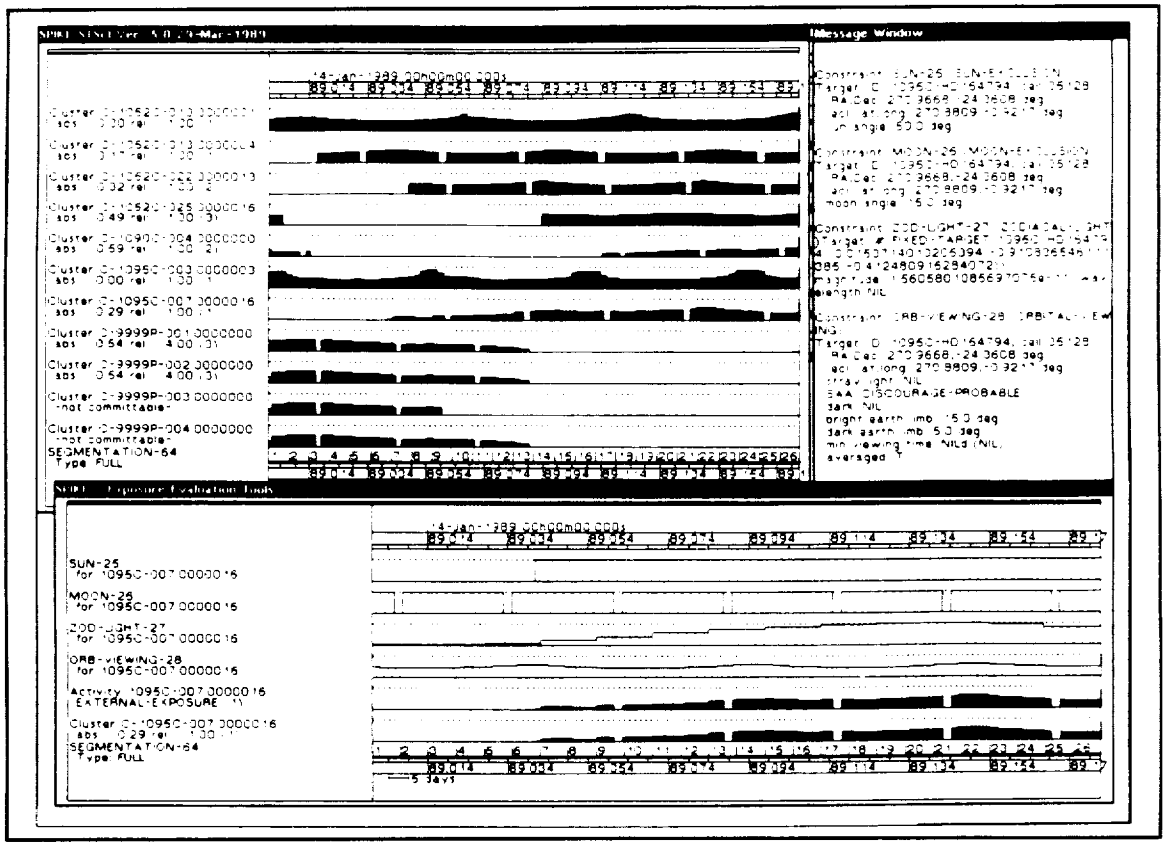
SPIKE also keeps track of *resource constraints* such as available data storage, available exposure time, available TDRSS down-link time, and so on. Each resource is represented as a suitability function that will reflect lower suitability as the resource is consumed in a given time segment.

The mode of SPIKE usage considered in this paper is long range scheduling. In this mode, the overall scheduling interval is divided into discrete units called *segments*. The length of a segment is arbitrary but expected to be one week during normal operations.

A long range schedule will consist of a number of time segments each of which will have a set of scheduling clusters that have been committed there. The commitments are to week-long segments and do not specify precise times. Periodically, the information about one segment will be communicated to SPSS which will then build a more detailed schedule. The logic of this organization is based on the notion that SPIKE can attempt to optimize a year-long schedule. SPSS will then have far fewer SUs upon which to work and will operate in much less time. Scheduling the SUs in that time range should be successful (based on SPIKE calculations), and a higher quality schedule will result.

Figure 1 illustrates the graphical interface to the SPIKE system.

ORIGINAL PAGE IS
OF POOR QUALITY


SPIKE SENSE Ver 5.0 29-Mar-1989
Message Window
SPIKE Exposure Evaluation Tools
14-Jan-1989 00h00m00.000s
89.014
89.034
89.054
89.074
89.094
89.114
89.134
89.154
89.174
5 days

```
Cluster C-1052C-010 0000001
 abs 0.00 rel 1.00 1
Cluster C-1052C-013 0000004
 abs 0.17 rel 1.00 1
Cluster C-1052C-022 0000013
 abs 0.32 rel 1.00 2
Cluster C-1052C-025 0000016
 abs 0.49 rel 1.00 (3)
Cluster C-1090C-004 0000000
 abs 0.59 rel 1.00 2
Cluster C-1095C-003 0000003
 abs 0.00 rel 1.00 1
Cluster C-1095C-007 0000016
 abs 0.29 rel 1.00 1
Cluster C-9999P-001 0000000
 abs 0.54 rel 4.00 (3)
Cluster C-9999P-002 0000000
 abs 0.54 rel 4.00 (3)
Cluster C-9999P-003 0000000
 -not committable-
Cluster C-9999P-004 0000000
 -not committable-
SEGMENTATION-64
 Type FULL
```

```
Constraint SUN-25 (SUN-EXCLUSION)
Target ID 1095C-HD164794 (all 05128
  RA,Dec 270.9668,-24.3608 deg
  ecl lat,long 270.8809,-0.9217 deg
  sun angle 50.0 deg

Constraint MOON-26 (MOON-EXCLUSION)
Target ID 1095C-HD164794 (all 05128
  RA,Dec 270.9668,-24.3608 deg
  ecl lat,long 270.8809,-0.9217 deg
  moon angle 15.0 deg

Constraint ZOD-LIGHT-27 (ZODIACAL-LIGHT)
Target # FIXED-TARGET 1095C-HD16479
4 0.0153714010205394 -0.910832654...
085 -0.4124809152840721)
magnitude 1.5605801085697075e-11 wav
elength NIL

Constraint ORB-VIEWING-28 (ORBITAL-VIEW
ING)
Target ID 1095C-HD164794 (all 05128
  RA,Dec 270.9668,-24.3608 deg
  ecl lat,long 270.8809,-0.9217 deg
  stray light NIL
  SAA DISCOURAGE-PROBABLE
  dark NIL
  bright earth limb 15.0 deg
  dark earth limb 5.0 deg
  min viewing time NIL0 (NIL)
  averaged T
```

```
SUN-25
 for 1095C-007 0000016
MOON-26
 for 1095C-007 0000016
ZOD-LIGHT-27
 for 1095C-007 0000016
ORB-VIEWING-28
 for 1095C-007 0000016
Activity 1095C-007 0000016
 EXTERNAL-EXPOSURE (1)
Cluster C-1095C-007 0000016
 abs 0.29 rel 1.00 (1)
SEGMENTATION-64
 Type FULL
```

Figure 1. Example screen from SPIKE showing the scheduling of a few HST observations. The upper left window represents a six-month scheduling interval and includes only scheduling clusters. The bottom window shows one of those clusters, its component activity and constraints. The upper right window displays textual information about several of the constraints.

1.3.2. Neural Network Schedule Optimizing System

The scheduling constraint satisfaction problem (CSP) can be represented as a Hopfield discrete neural network (Johnston, 1989) which can be thought of as a matrix where the rows represent scheduling clusters and the columns represent discrete time segments. The output state of each neuron in the matrix can be either 0 or 1 where 1 indicates a commitment of the activity to the time segment. A column of guard cells is used to bias the network in such a way as to maximize the number of clusters that are scheduled (neurons that are on).

A congruous connection matrix stores the connections between neurons representing relative constraints. Those *connections* are derived by analyzing the effect of committing an activity A to a time segment S on all other possible activity/segment neurons in the matrix. Another matrix stores the *biases* associated with each neuron. These biases are assigned by analyzing the absolute constraints on activities that comprise clusters, in such a way that higher biases indicate greater suitability.

The term *summed suitability* is defined to be a function of the sum of all inputs to each neuron that is on in the network. This is one way to measure how good the overall schedule is.

1.4. Interactions between SPIKE and SPSS

SPIKE to SPSS communication

SPIKE sends information to SPSS concerning what scheduling units are to be placed in a specific week. The granularity is very coarse in that SPIKE assures SPSS that the SU is schedulable at some point in that week (but not at which point). The operators of SPSS then must attempt to place each SU onto a detailed timeline.

SPSS to SPIKE communication

It may be the case that SPSS will be unable to place certain SUs on its timeline. The reasons for this might include:

1. The philosophy of scheduling at STScI includes the provision for an **oversubscription** of exposures that is the rule rather than the exception. The SPIKE system therefore has adjusted the appropriate parameters associated with various resource constraints (exposure time, data volume) such that 30% oversubscription is the scheduling goal.

2. Since SPIKE is a long term scheduler, some constraints (e.g., South Atlantic Anomaly of the Van Allen Belts, TDRSS satellite availability) must be calculated only on a statistical basis. This is due to the fact that the ST in-track position is not accurately predictable on long time scales.

3. The logical context may have changed from the time that SPIKE calculated its best schedule to the time when SPSS attempts to place SUs on its calendar.

    - Minor changes in the orbit model may invalidate certain SPIKE decisions. For example, solar activity may have changed unpredictably such that constraints based on such activity (e.g., SAA) become more severe at SPSS scheduling time.

    - On-orbit experience with the spacecraft may change the manner in which activities are scheduled.

4. The greedy algorithm employed in the auto-scheduling SPSS subsystem may select from the search space a set of SU/time assignments such that certain mutually conflicting constraints make a complete scheduling of all SUs assigned to the week impossible.

It is a possibility that the execution by the spacecraft of certain exposures may fail (or that the data resulting from an exposure may be lost). In such a case, the SPIKE system will have to be alerted to this partial failure. This may require the creation of a new SU consisting of only the activities affected that will have to be considered for rescheduling.

In the event that SPSS is unable to schedule a subset of the SUs for a week, information concerning that subset will be sent to the SPIKE system along with some rudimentary explanation information (e.g., "constraint C violated", "instrument X unavailable"). This new information will, of course, make invalid portions of the SPIKE schedule. The corresponding scheduling clusters will have to be removed from the schedule and reprocessed. Those steps are the focus of this discussion.

1.5. CCOPS

A SPIKE subsystem called Constraint Cascading Over Planning Sessions (CCOPS) has been developed to do the following:

CCOPS facilitates the decomposition of the full scheduling problem into more manageable portions. If the SPIKE system is called upon to build a schedule and all proposals are loaded at once, about 15,000 activities may be memory resident. As the number of activities and constraints increases, the time required to load files and instantiate the database and the time required for computation will increase. If it is the case that all activities and constraints are memory-resident at one time and a complete schedule has been computed, the problems associated with a hardware or software crash are exacerbated because a major loss is sustained. The goal then has been to break the pool of proposals into groups.

The CCOPS system processes *session monitors* which retain an abstract memory about what proposals are grouped, what scheduling decisions have been made, and what resources have been consumed. The CCOPS system interface is menu-oriented and provides the user with tools to group proposals. Each group can be loaded into and processed by the SPIKE scheduling tools. Scheduling decisions made are stored by CCOPS in a symbolic format in a database that can be saved to disk. The important feature provided by CCOPS is a protocol for communicating the consumption of resource from one group of proposals to another.

The CCOPS system helps to solve the rescheduling problem by providing a mechanism for dealing with first the high priority items followed by the supplemental ones since, at the very least, the pool will be divided by director's priority.

Let SM_a and SM_b be session monitors that are ordered (i.e., $a < b$). The CCOPS system supports *constraint cascading* where information about the resources consumed by SM_a is communicated to SM_b. The cascade is unidirectional and so no information may be passed from SM_a to SM_b. Thus, it is important that SM_a be fully scheduled before any scheduling is done in SM_b. Otherwise incorrect scheduling decisions would be made. Similarly, rescheduling of clusters assigned to SM_a should be effected before those assigned to SM_b.

2. Functions used to quantify the problem

An important component of the SPIKE scheduling methodology is based on the notion that constraint information (e.g., "schedule A before B") can be represented numerically as suitability functions. In that spirit, two new measures, *rescheduling priority* and *disruptivity,* are proposed. These functions map preferences related to rescheduling into numeric values so that they can be considered along with other constraints and preferences.

2.1. Rescheduling Priority

Given a specific scheduling cluster , it is desirable to deduce a numeric preference that can be used via comparison to select such objects for rescheduling.

The term *rescheduling priority* is defined to be the relative measure of how important it is to reschedule a cluster. This priority is a single numeric value that is determined in the following manner. Each element in a set of criteria is considered. With respect to a specific criterion, the cluster is analyzed, yielding a numeric value, the *sub-priority*. All such values are multiplied producing the rescheduling priority. The behavior of the multiply function is such that if any value (determined for a specific criterion) is zero then the rescheduling priority is zero. Each analysis is therefore done with that fact in mind.

The criteria that are proposed for consideration are described below.

2.1.1. Partial scheduling of a dependency cluster

In certain cases, it may be that the activities in a proper subset of the scheduling clusters in a dependency cluster are not scheduled. The numeric value associated with such a case is calculated in the following manner: Let C be the number of clusters in a dependency cluster, S_u be the number of unscheduled clusters, and S_s be the number of scheduled clusters. The priority is the ratio S_s/C. This is a subjective measure based on the notion that a dependency cluster that has a higher percentage of scheduled clusters (and thus is closer to being completely scheduled) ought to be processed before a cluster with a lower percentage. The activities in a dependency cluster are linked via constraints and therefore represent a scientific experiment.

2.1.2. Partial scheduling of a Proposal

Similar logic utilized in the case of the dependency cluster can be applied to the set of activities in a proposal. Here the scientific value of completion may be even stronger. If C is the number of activities in a proposal, then the same ratio S_s/C is used to find the numeric value for this preference.

2.1.3. Director's Priority

Each proposal has an assigned *director's priority* which is one of **high** or **supplemental**. Using this information, the sub-priority of a cluster is 1 if the director's priority (of the source proposal) is high and 0 otherwise. The pool of supplementals is large, the current philosophy states that supplemental proposals are not guaranteed scheduling, and so this criterion will give supplementals originally not scheduled by SPIKE a chance.

2.1.4. Priority based on repeated SU failures

It is possible that a specific SU will repeatedly fail to be scheduled by SPSS. One reason for this is the oversubscription philosophy mentioned above. It is proposed that a priority value be calculated to capture those iterations for use by the SPIKE rescheduling machinery. If N is the number of times that SPSS has rejected a specific SU, then the *repeat failure priority* is 1/N (unless N >= **threshold** in which case it is 0). The threshold is currently assigned the value three. For example, if the SU has been rejected by SPSS 2 times then its priority is 1/2.

2.1.5. Some clusters cannot be rescheduled

If C_i is a cluster to be rescheduled, other components of the dependency cluster (to which C_i belongs) have been either executed by ST or have been scheduled in the very near term (and hence may not be unscheduled), and there is no suitable time segment for C_i due to constraints then it is impossible to reschedule C_i and its priority must be zero.

2.2. Disruptivity

The function *SD(scheduling cluster, time segment)*, the suitability based on disruptivity, is defined to be a relative measure of the effects of scheduling *scheduling cluster* in *time segment*. Such changes would include other clusters being uprooted and rescheduled. Unlike the rescheduling priority, sd takes the form of the classic suitability function. An SD of one means that little if any disruption is expected. An SD of zero means that an unacceptably high disruption is predicted.

Disruptivity can be calculated by taking the following factors into account:

2.2.1. Disruptivity and estimated propagated effects

If one reschedules an activity, what are the effects of that on the other activities in the dependency cluster of the activity? The best case scenario would be if no other activities must be moved from their positions on the pre-existing schedule. Assuming that one is able to get all other activities back on the schedule, the worst case exists when all other activities in the cluster must be shuffled within the schedule in order to accommodate the activity.

The other important criterion related to determining SD is what happens to the overall suitability of the schedule. The suitability of the old (and now invalid) schedule is the baseline. If the suitability of the new schedule increases or remains constant, then disruption is low and sd is close to one. If the suitability decreases, the SD is less than one.

2.2.2. Resource Consumption and Disruptivity

Determining the overall summed suitability of a schedule can be used to determine how placing one activity will affect resource consumption. The neural network system maintains a network that encodes how a specific scheduling decision affects other activities based on resource consumed. If a decision is made that causes many activities to become unschedulable based on available resource then this will be included in the calculation to produce a relatively higher disruptivity.

3. Two Algorithms for Rescheduling an Activity

In the following paragraphs, two algorithms that can be used to solve the *single activity rescheduling problem* are proposed. In both algorithms schedule time is divided, by the concept of *now* (some time segment), into the *past* (all segments lower in ordinality than now), and the *future* (all segments higher than but including now). The selection of now should represent not real clock time but instead the point in the real time future that is where one can reasonably make changes to the schedule. For instance, one probably would not want to routinely make changes to a schedule for the next week in real time. However, making changes two months in the future might be acceptable. Therefore, assigning now to be a month into the real time future is reasonable.

The first step in either algorithm is to order the list of scheduling clusters to reschedule based on their relative rescheduling priority (see above). The following discussions relate only to rescheduling a single cluster that is selected from such a list.

In both cases disruptivity is calculated in the following manner. If any unschedulables are noted then disruptivity is 1. Otherwise the disruptivity is percent of activities that moved. The suitability based on disruptivity is of course 1 - disruptivity.

3.1. A Stochastic NN Algorithm

The stochastic NN rescheduling algorithm is based primarily on the notion that enough intrinsic knowledge of the clusters and constraints is stored in the biases and weights of the neural network system to quickly (and optimally) replace an activity on a pre-existing schedule. Let A be the cluster that is to be rescheduled. The steps are described below.

1. Freeze the past.

    a. Turn on the neurons in the past that represent accepted commitments (of clusters to segments). Clamp those neurons (with a high bias) so that their state can not change.

    b. For rows that are in the past that have no neurons on, clamp all neurons in those rows with a negative bias to prevent any neurons there from being turned on.

2. Eliminate the original commitment (of the cluster to be rescheduled) from the range of commitment possibilities by turning the corresponding neuron off and clamping it with a large negative bias.

3. Turn on all neurons in the future that correspond to legal commitments. The assumption here is that these pre-existing commitments are valid and represent a baseline schedule. These neurons are **not** clamped using bias and so during a network run may change state. The underlying logic is this: It is desirable to preserve as much of the pre-existing schedule as possible. However, no scheduling decision that lies in the future cannot be revoked in order to reschedule A. Figure 2 illustrates an example neural network representation at this point.

4. Run the neural network scheduler. Since the clusters in the dependency cluster of A except for A are scheduled their weighted affects on the network will tend to place A in a legal place (that should be very close in time to its original placement). If such a legal place does not exist (because it is in the past), then some portion (perhaps all) of the dependency cluster must be moved. The more activities that are moved in order to accommodate the rescheduling of A the more the solution violates our goal of minimized disruptivity.

5. Given a solution determined in step 4, calculate the measure of disruption that has occurred.

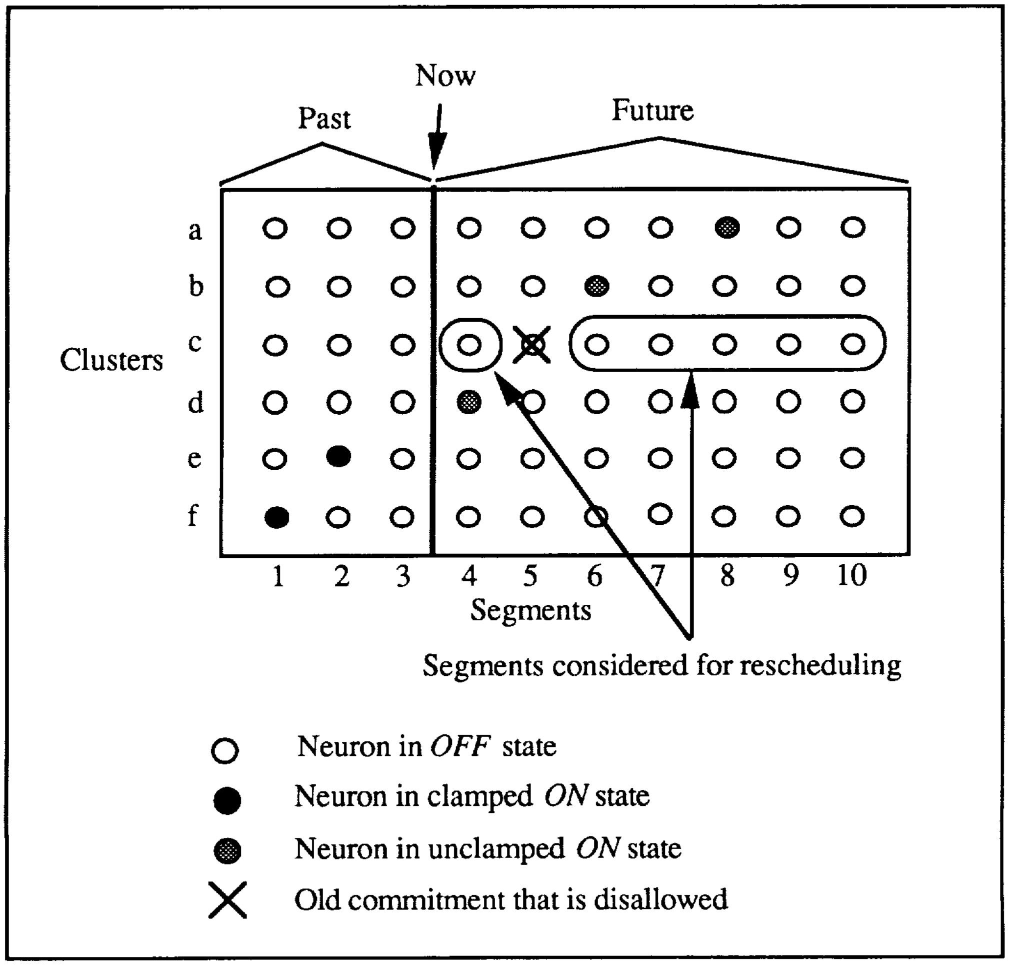

Now
Past
Future
a
b
c
d
e
f
Clusters
1 2 3 4 5 6 7 8 9 10
Segments
Segments considered for rescheduling
○ Neuron in *OFF* state
● Neuron in clamped *ON* state
◍ Neuron in unclamped *ON* state
✕ Old commitment that is disallowed

Figure 2. Neural network representation of a schedule. Each circle in the figure depicts a neuron (representing a possible commitment). Cluster c, in this scenario, had previously been committed to segment 5. That commitment is now illegal. Black circles represent commitments that are considered unchangeable (because they are in the past). Gray circles represent commitments in the schedule that can be uncommitted (in order to reschedule cluster c). Both algorithms begin by creating a network that has this general organization.

3.2. An Exhaustive Algorithm to Minimize Disruptivity

In this algorithm, the neural network environment and external functions are also used to determine disruptivity. However, in this algorithm an exhaustive search is effected to determine the best possible place(s) where cluster A can be rescheduled. The method used to estimate disruptivity will operate in the following manner:

1. Freeze the past, and turn on legal commitments in the future (same as above).

2. For each time segment in the future (except for the disallowed segment), find the disruptivity that results from scheduling A there.

   a. First turn on the corresponding neuron and set its bias high.

b. Run the network. The network may cause clusters in the future to move because the immutable commitment of A may cause certain commitments to be inconsistent (based on the weights and biases).

c. Analyze the disruption.

3. The segment that produces the best (lowest) disruptivity is selected as the place for rescheduling A. In the case of ties, the earliest segment is selected.

Informing SPIKE about disruptivity.

Once the suitability of disruptivity has been calculated for a given activity, it may be useful to communicate that inferred knowledge to the core SPIKE scheduling system. The suitability can be integrated into the planning session data structures as an *absolute time constraint*, represented graphically for users, and may guide automatic or manual rescheduling of activities.

4. Prototype systems and experimental results

We have implemented a prototype based on the discussion above in order to test whether the behavior of the exhaustive rescheduling algorithm justifies the computational expense relative to the stochastic neural network rescheduling approach.

Hypotheses

Let $\mu 1$ be the mean disruptivity of the neural network approach and $\mu 2$ be the mean disruptivity of the exhaustive algorithm.

H_0: $\mu 1 = \mu 2$

H_1: $\mu 1 \neq \mu 2$

In a setup that was composed of 60 segments, 30 scheduling clusters, and dependency clusters of size 3, 50 trials were run for both the network rescheduling algorithm and the exhaustive rescheduling algorithm to determine which would find the best place to reschedule such that disruptivity was minimal. For the network rescheduling algorithm the mean was 0.139 and the standard deviation was 0.318. For the exhaustive rescheduling algorithm, the mean was 0.038 and the standard deviation was 0.141. A 95% confidence interval (0.005, 0.197) for the difference in population mean scores was determined using the *Z statistic* (Bhattacharyya, 1977). Since the interval does not include zero, the null hypothesis is rejected in favor of H_1.

5. Discussion

The exhaustive approach to rescheduling appears to generally produce better results. Statistics reveal that the differences however are not great and so one might argue that the computation involved in the exhaustive approach is too costly given the marginal benefit. Although a large (50) number of trials were executed, the algorithms were only tested on a single problem. More testing on a varied set of problems is required in order to more accurately assess the comparative usefulness of these approaches. It is also possible that the selected parameters (e.g., number of links, position in time of the rejected cluster) may have biased the results. Again, only more tests will tell.

It is considered odd that the exhaustive system was only a little better statistically. First, one may argue that statistical measures are designed to be conservative with respect to supporting differences that result from varying treatments. Another important point is that, in general, the best place to reschedule a cluster is another point in time that is close to the original segment. If such a place exists that is legal (based on constraints), then the stochastic algorithm should find this solution. It is when that nearby place is not legal that the exhaustive algorithm should prevail because the stochastic algorithm will then find any legal configuration without regard for disruptivity.

It is believed that the basic approach described in this report is sound and when fully implemented will provide an effective mechanism to repair broken schedules when that need arises.

Acknowledgements

The authors would like to thank the following persons for their ideas concerning this report: Dr. Glenn Miller and Shon Vick.

References

Bhattacharyya, G., and Johnson, R. (1977). *Statistical Concepts and Methods.* New York: John Wiley & Sons, Pub.

Garey, M., and Johnson, D. (1979). *Computers and Intractability.* New York: W.H. Freeman and Co.

Hall, D., ed. (1982), *The Space Telescope Observatory*, NASA CP2244.

Miller, G., Johnston, M., Vick, S., Sponsler, J., and Lindenmayer, K. (1988). Knowledge Based Tools for Hubble Space Telescope Planning and Scheduling: Constraints and Strategies, in *Proc. 1988 Goddard Conference on Space Applications of Artificial Intelligence*; reprinted in *Telematics and Informatics* 5, p. 197 (1988).

Johnston, M., Adorf, Hans-Martin (1989). Learning in Stochastic Neural Networks for Constraint Satisfaction Problems. In *Proc. NASA Conf. on Space Telerobotics.*

Johnston, M. (1990). SPIKE: AI Scheduling for NASA's Hubble Space Telescope. To appear in *Proc of the Sixth IEEE Conference on Artificial Intelligence Applications* (March, 1990).

N90-22300

SPACE COMMUNICATIONS SCHEDULER: A RULE-BASED APPROACH TO ADAPTIVE DEADLINE SCHEDULING

Nicholas Straguzzi
GE Advanced Technology Laboratories
Moorestown, NJ 08057

ABSTRACT

Job scheduling is a deceptively complex subfield of computer science. The highly combinatorial nature of the problem, which is NP-complete in nearly all cases, requires a scheduling program to intelligently traverse an immense search tree to create the best possible schedule in a minimal amount of time. In addition, the program must continually make adjustments to the initial schedule when faced with last-minute user requests, cancellations, unexpected device failures, etc. A good scheduler must be quick, flexible, and efficient, even at the expense of generating slightly less-than-optimal schedules.

The Space Communications Scheduler (SCS) is an intelligent rule-based scheduling system developed at GE's Advanced Technology Laboratories. SCS is an adaptive deadline scheduler which allocates modular communications resources to meet an ordered set of user-specified job requests on board the NASA Space Station. SCS uses pattern-matching techniques to detect potential conflicts within a schedule, then resolves these conflicts through algorithmic and heuristic means. As a result, the system generates and maintains high-density schedules without relying heavily on backtracking or blind search techniques. SCS was designed to allocate communication devices on board the Space Station, but its general-purpose scheduling strategy is suitable for many common real-world applications.

1.0 INTRODUCTION

"Scheduling" is a term very familiar to most people. Personal schedules are routinely made and revised; it is a task so common that few people think about the cognitive actions involved in its performance. Yet, the seemingly simple act of scheduling, which can be loosely defined as allocating the resources necessary to perform a set of jobs over a specific time interval, is one of the more complex problem areas of computer science.

Two general classes of scheduling problems exist: *precedence constrained* and *deadline*. Precedence constrained scheduling (sometimes called simply constraint scheduling) is very closely related to classical computer planning problems. In its most basic form, constraint scheduling generates an agenda for performing subtasks of a specified job, given a partial ordering of the subtasks and a deadline for completing the job. Garey and Johnson (Garey and Johnson, 1979) illustrate constraint scheduling with the example of a college freshman building a four-year course plan. Because certain courses are required for graduation, and because most of these courses have prerequisites of their own, developing an appropriate schedule is, as every college student knows, a non-trivial task.

Deadline scheduling (sometimes called interval, appointment, or timetable scheduling) is a somewhat more familiar problem class. Here, the goal is to create an optimal timetable for the execution of a set of jobs over a specific interval of time, given a finite set of available resources and a set of acceptable release times (earliest start times), deadlines (latest completion times), and priorities for each job. Common real-world examples include a doctor's receptionist scheduling patient appointments and a computer's operating system scheduling the execution of batch programs. What actually constitutes an "optimal" schedule varies from application to application. In the first example, an optimal schedule would be one that allows the maximum number of appointments, while in the second, it might maximize the sum of the priorities of the executed programs.[1]

[1] Many application areas fall into both scheduling classes. Engineers at a car manufacturing plant, for example, must make use of both constraint scheduling (deciding the optimal order for assembling the parts of a car) and deadline scheduling (allocating the personnel and resources to perform each task at the appropriate time).

Both constraint and deadline scheduling are NP-complete in virtually all non-trivial cases (Garey and Johnson, 1979), which forces all automated scheduling systems to rely heavily on heuristic search techniques. Unfortunately, certain real-world scheduling considerations make good schedules extremely difficult to generate, even heuristically. The resources available to perform the jobs may be very limited or they may not be shareable between jobs. Jobs may have varying durations or variable release times, they may be uninterruptible, or they may not be permitted to run concurrently with other jobs. In addition, a scheduler is not necessarily finished after the initial schedule is made. Unforseen circumstances often arise during job execution time, such as a last-minute emergency request or an unanticipated equipment failure, that require the scheduler to make "on-the-fly" adjustments.

1.1 Related Research

Because of the very diverse nature of scheduling problems, as well as their inherent intractability, the goal of computer science researchers is not to create one general-purpose program that can handle every conceivable scheduling problem, nor to create a program that guarantees optimal schedules instantaneously. Rather, the goal is to create programs that generate near-optimal schedules, in a reasonable amount of time, for one specific subclass of scheduling problems.

Most recent research has concentrated on Job-Shop Scheduling (JSS), a general subclass of problems within the domain of Operations Research (Martin and Pling, 1978; Marcus, 1984). The goal of most JSS systems is to develop near-optimal schedules for manufacturing or industrial facilities where slight improvements in scheduling efficiency may translate into huge amounts of savings to a company. Another popular research area is the development of schedulers for computer operating systems (OS) (Deitel 1984; Tanenbaum, 1987). Many OS textbooks include a chapter on scheduling; Deitel's *An Introduction to Operating Systems* (Deitel 1984) contains a good set of goals for an OS scheduler.

Within the field of Artificial Intelligence (AI), scheduling is part of the Planning Systems domain (Nillson, 1980). Much of this research is concerned with modeling the real-world planning environment and representing the effects that certain actions have on the model. Deadline scheduling is often represented as the lowest level on the planning tree, performed only after goals are identified and task sequences are determined. Sophisticated planning systems will consider deadline scheduling restrictions as part of the overall plan generation process (Hayes-Roth et al., 1979).

Fox and Kempf (1985) have studied the dual problems of computational complexity and executional uncertainty on job-shop scheduling domains. From this, they have defined two basic principles for building an efficient scheduler. The "Principle of Least Commitment" states that a scheduler should never commit a job to a specific time interval or resource set until there is a good reason to do so. The "Principle of Opportunism" states that a scheduler should take advantage of all available opportunities to reduce its search space.

1.2 Terminology

A *job* (also called a *service*) is a single, indivisible, real-world task to be performed within a specified time interval. The actual nature of a job varies from application to application. To a doctor, a job may be one consultation session with a patient. To a factory line worker, a job may be assembling ten electric motors by a certain deadline. Associated with each job are a *priority*, a set of *preconditions* that must be met before the job may begin (also expressible as *set-up time*), the time *constraints* for scheduling the job, and a set of *resources* needed to perform the job.

A *resource* is anyone or anything available for use in the execution of a job, such as a person, a work area, a tool, or a raw material. Like jobs, resources are assumed to be indivisible units for scheduling purposes. The maximum number of jobs that a resource can support at one time is known as its *capacity*. A *dedicated resource* has a capacity of one, while a *shareable resource* has a capacity greater than one.[2]

[2] Note that *shareability* and *indivisibility* are not mutually exclusive terms. A mainframe computer is *shareable* in the sense that more than one user may be logged in at any given time. It is *indivisible* in that a user does not request the "left half" or the "bottom one-third" of the computer when reserving CPU time. Similarly, a box of ten identical screwdrivers may be thought of as ONE shareable, indivisible resource with a capacity of TEN jobs.

The number of jobs actively being supported by a resource at any given time is known as the resource's *load*. An *idle* resource is one with a load equal to zero. A *free* (or *available*) resource has a load less than its capacity, while a *busy* resource has a load equal to its capacity. An *overloaded* resource has a load greater than its capacity and indicates that an error condition is present in a schedule. The jobs competing for an overloaded resource are said to be in *conflict*.

A *scheduler* (whether human or machine) takes a description of the set of jobs to be performed and the resources available to perform them, and produces a schedule which maps resources to jobs. An *allocation* is when one resource is reserved for one job over one interval of time, and a *supported* job is one which has reserved all of the resources necessary for its execution. Finally, a *schedule* is any mapping of resources to jobs.

2.0 SPACE STATION COMMUNICATIONS

GE's Government Communications System Division (GCSD) is a member of the McDonnell-Douglas team awarded NASA's Space Station Work Package II. GCSD's task is to develop the Space Station's Communications and Tracking System (C&TS).

C&TS will be comprised of a number of subsystems, each handling a specific class of communications (see Figure 2-1). The Space-To-Space Communications (STSC) Subsystem, for example, supports links between the Space Station and non-terrestrial sources (satellites, the Space Shuttle, etc.). All subsystems consist of a set of modular communications devices that can be configured in a variety of ways, depending on the Space Station's current needs. A set of devices that supports a single communication link is called a *string*; at any given time, a subsystem may have several (or zero) active strings.

All C&TS subsystems are managed by the Control and Monitoring (C&M) Subsystem, which is responsible for allocating communications resources, monitoring the performance of on-line devices, diagnosing equipment failures, and taking whatever actions are necessary to maintain error-free communication links. GE engineers, as part of an ongoing IR&D project, have been eval-

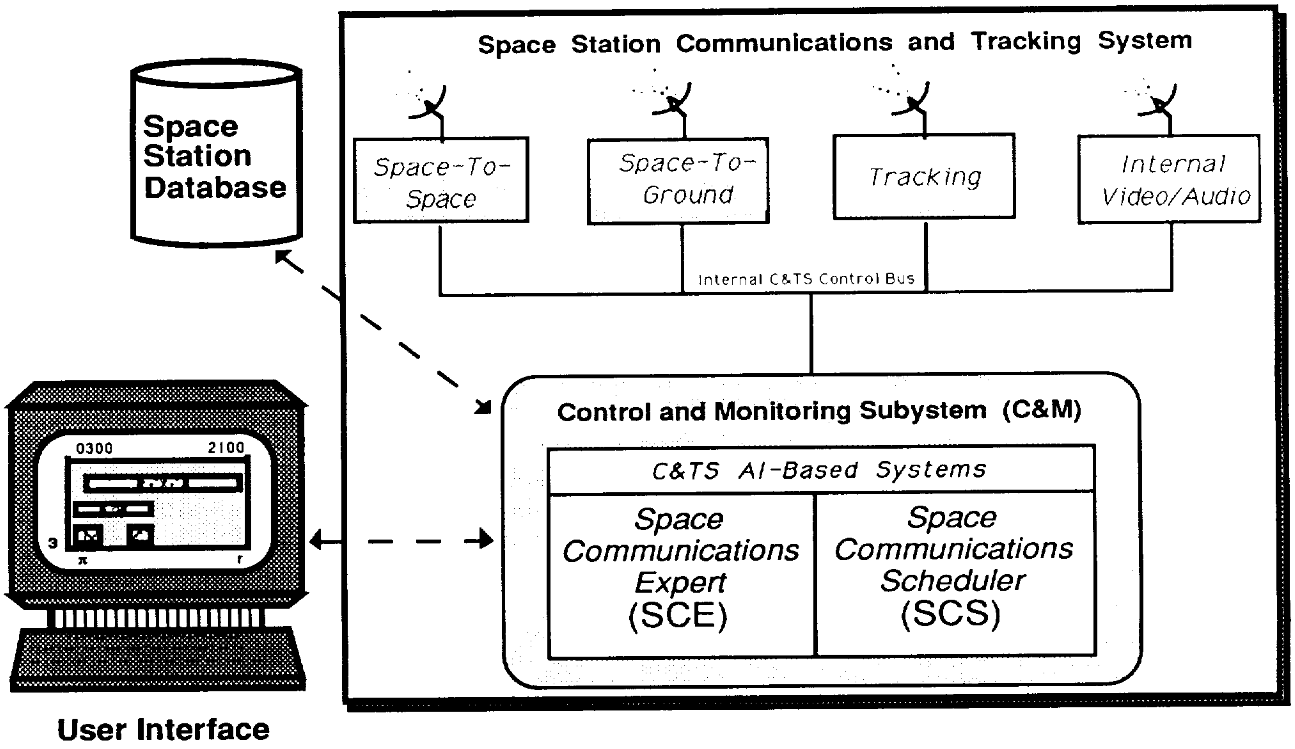

Space Station Communications and Tracking System
Space Station Database
Space-To-Space
Space-To-Ground
Tracking
Internal Video/Audio
Internal C&TS Control Bus
Control and Monitoring Subystem (C&M)
C&TS AI-Based Systems
Space Communications Expert (SCE)
Space Communications Scheduler (SCS)
0300
2100
3
User Interface

Figure 2-1. Architecture of Communications and Tracking System (C&TS).

uating the feasibility of integrating two knowledge-based systems within C&M: the Space Communications Expert (SCE) and the Space Communications Scheduler (SCS).

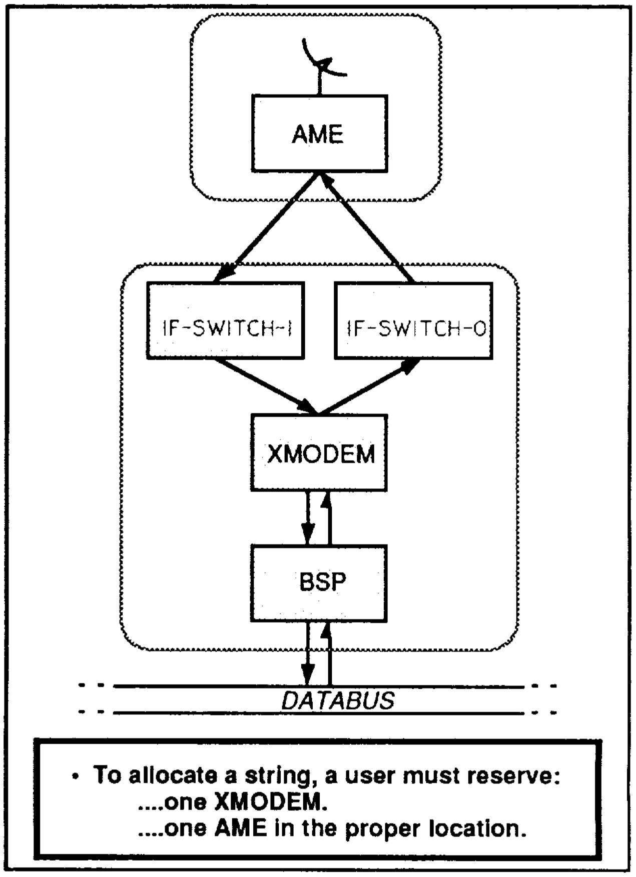
GE's Advanced Technology Laboratories (ATL) developed SCE in 1987 as an embedded expert system designed to monitor and maintain strings within the STSC Subsystem. SCE allocates new strings on command, then evaluates data from various sources, such as external test procedures, device status reports, and the global Space Station database, to ensure that the string is operating normally. When anomalies are detected in a communication link (comlink), SCE isolates and replaces the device causing the problem. In 1988, GCSD developed an expanded version of SCE, the Prototype Intelligent Space Communications Expert System (PISCES). PISCES extends SCE to include both strings and partial device failures in the Space-To-Ground Communications (STGC) Subsystem.

SCS and PISCES were conceived as cooperating expert systems that form the "brain" of C&M. SCS's role is to allocate and schedule C&TS devices, and then transfer control to PISCES which assembles and maintains the resulting strings. When PISCES recognizes a device failure, it notifies SCS, which in turn adjusts its schedule accordingly and specifies an available replacement device to PISCES.

2.1 SCS Scheduling Domain

C&TS is a relatively standard deadline scheduling domain in which "jobs" correspond to individual communication links (called "services"), and "resources" are the modular communication devices used to create strings (transceiver-modems, switches, fiber-optic links, antennas, etc.).

As with SCE, the first-year development effort of SCS concentrated solely on the STSC Subsystem. A string within STSC typically consists of five interconnected devices (see Figure 2-2). Transmitted signals first travel through a Baseband Signal Processor (BSP) which connects STSC with the many data busses on board the Space Station. The Transceiver-Modem (XMODEM) modulates this signal and sends it through an outgoing Intermediate-Frequency Switch (IF-SWITCH) to a specific Antenna Mounted Equipment (AME) from which it is transmitted. Received signals traverse the same path in the opposite direction, except that an incoming IF-SWITCH is used.

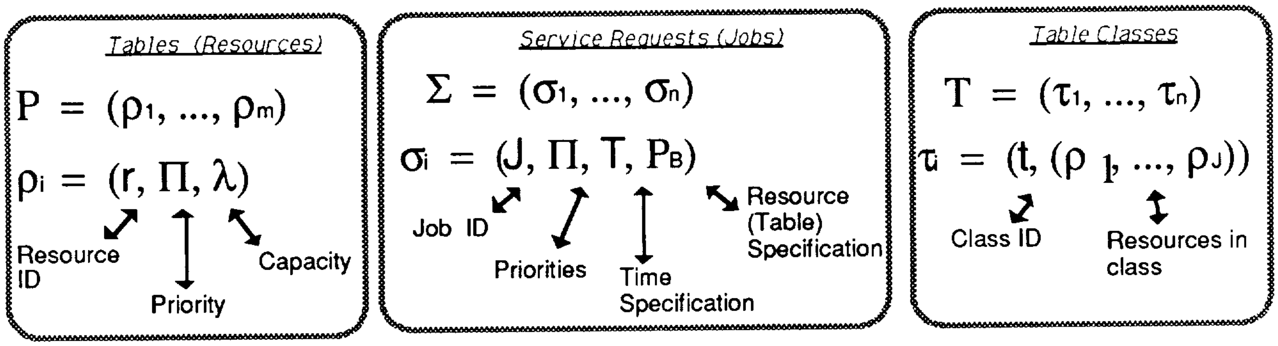


- To allocate a string, a user must reserve:
  ....one XMODEM.
  ....one AME in the proper location.

Figure 2-2. Standard Space-To-Space Communications String.

The two critical devices on an STSC string are the Transceiver-Modem and the Antenna. Selecting an XMODEM forces the selection of the BSP and IF-SWITCHes because they are hardwired together. The four types of AMEs are OMNI, AIR-LOCK, SERVICE-BAY, and PARABOLIC. They are located at various fixed spots on the outside of the Space Station. To allocate an STSC string, one must allocate an XMODEM (any operational one will do) and an AME of the appropriate type in the appropriate location.

Figure 2-3 shows the XMODEMs and AMEs of STSC represented as SCS tables.



P =

ρ1 = (XMODEM-A1, 100, 1)
ρ2 = (XMODEM-A2, 100, 1)
ρ3 = (XMODEM-A3, 100, 1)
ρ4 = (XMODEM-B1, 100, 1)
ρ5 = (XMODEM-B2, 100, 1)
ρ6 = (XMODEM-B3, 100, 1)
ρ7 = (OMNI-AME-1, 80, 3)
ρ8 = (OMNI-AME-2, 80, 3)
ρ9 = (OMNI-AME-3, 80, 3)
ρ10 = (OMNI-AME-4, 80, 3)
ρ11 = (AIR-LOCK-AME-1, 80, 3)
ρ12 = (AIR-LOCK-AME-2, 80, 3)
ρ13 = (GIMBAL-AME-1, 80, 1)
ρ14 = (GIMBAL-AME-1, 80, 1)
ρ15 = (SERVICE-AME-1, 80, 3)

T =

τ1 = (XMODEM,
(XMODEM-A1, XMODEM-A2,
XMODEM-A3, XMODEM-B1,
XMODEM-B2, XMODEM-B3))
τ2 = (OMNI-AME-PORT,
(OMNI-AME-1, OMNI-AME-3))
τ3 = (OMNI-AME-STARBOARD,
(OMNI-AME-2, OMNI-AME-4))

Figure 2-3. Resource tables and table classes for C&TS.

3.0 SPACE COMMUNICATIONS SCHEDULER

SCS is a rule-based scheduling system developed by GE's Artificial Intelligence Laboratory in Moorestown, NJ. SCS was designed to allocate and schedule modular communications equipment on board the NASA Space Station, automatically making adjustments during job execution time when faced with unexpected device failures or last-minute user requests. It combines algorithmic and heuristic search techniques, sophisticated pattern-matching capabilities, and a flexible scheduling strategy adaptable to many different applications. SCS was implemented using Inference's Automated Reasoning Tool (ART™) expert system shell augmented with custom LISP and C code; it runs on a Digital™ VAX computer.

SCS addresses deadline scheduling problems char- acterized by:

1. *Continuous, indivisible jobs* — Once started, a job will not be preempted before completion.

2. *Negligible or constant set-up times between jobs* — Set-up times are usually a trinary function of two jobs and one resource, yielding a time. For example, **SETUPTIME(A,B,R) = 10** means that it will take 10 minutes to "re-set" resource R between the end of job A and the start of job B. SCS requires that all set-up times are either negligible (in which case they can be ignored) or relatively constant (in which case they can be automatically added to the duration of each assignment).

3. *Low-capacity resources* — Each resource has a relatively low capacity, generally five jobs or less. SCS takes approximately 1.5 times longer to schedule a (λ+1)-capacity resource than a λ-capacity one.

4. *Partial order of job prioities* — Each job has its own scheduling priority, and no job may preclude a higher-priority job from being scheduled. In other words, it is better to schedule one job of priority 100 than 10 jobs of priority 90.

The subclass of scheduling problems handled by SCS is actually very common. Virtually any domain requiring "appointments" — from a doctor's office, to dinner reservations, to library books, to a college computer terminal room, to communications equipment aboard the Space Station — is a domain in which SCS is applicable.

3.1 SCS Data Structures

Within SCS, jobs are represented by *service-requests*, resources by *tables*, and allocations by *blocks*. Service-requests and tables, along with certain *scheduling parameters*, make up the input to SCS. No two service-requests or two blocks may be exactly alike. The lone output of the system are the blocks that make up the schedule. This section contains the formal definitions of the input and output specifications of SCS.

All SCS time specifications are in military format with discrete one-minute increments. A time interval is specified as an ordered pair (t_1, t_2), inclusive of its startpoint but not inclusive of its endpoint. For example, "(0800, 0900)" specifies 60 one-minute units of time, the first unit beginning at 8:00 AM and the 60th beginning at 8:59 AM. While this is a somewhat inelegant convention, no perfect way exists to model time as an ordered set of discrete elements (Allen, 1983).

3.1.1 Service-Requests

A service-request is a 4-tuple,

$$\sigma = (j, \pi, T, \rho)$$
$$\Sigma = \{\sigma_1, ..., \sigma_m\}$$
$$\mathbf{J} = \{\mathbf{j}(\sigma_i) \mid \mathbf{1 \leq i \leq m}\}$$

where j is the *job* represented by σ; π is an ordered pair, (π_p, π_i), specifying *priority*; T is a set of 4-tuples, $\mathbf{T = \{T_1, T_2, ...\}}$, $\mathbf{T_i = (t_\sigma, (\Delta_+, \Delta_-), MIN\Delta, d)}$ representing the *time constraints*; and ρ is the *resource set*, $(\rho_1, \rho_2, ...)$, required to perform s.

Each service-request corresponds to exactly one job, $j \in J$. However, a job may be represented by multiple service-requests, each with a different π, T, and/or ρ. Once a job is successfully scheduled by SCS, all alternative requests for that job are automatically deactivated.

π_p and π_i are called the *scheduling priority* and *inertia value* of σ. Scheduling priority is used only during the initial scheduling phase, and it represents the relative importance of σ in comparison to other service-requests. If and when a job is successfully scheduled, the inertia value specifies how difficult it is to "bump" that job during the rescheduling phase.

T, the time constraint for s, is itself a set of 4-tuples. Each $T_i \in T$ consists of a *start time* (t_s); the allowable negative and positive *offsets* from the start time, (Δ_-, Δ_+); a boolean flag (MINΔ) which, when set, specifies that the request should be scheduled as close as possible to t_s; and the job's *duration* (d). In standard terminology, the release time for σ is $(t_\sigma - \Delta_-)$ and the deadline for σ is $(t_\sigma + \Delta_+ + d)$. Each T_i in T signifies an equally acceptable time interval for scheduling the job.

Finally, ρ specifies the resource set for σ. Because resources are represented as "tables" within SCS, ρ is expressed as a nonempty set of table names or table classes (see Section 3.1.2). For σ to be successfully scheduled, all tables in ρ must be available at the specified time; otherwise, no resources are allocated and σ is deactivated.

An example of how to specify SCS service-requests is given in Figure 3-1.

3.1.2 Tables and Table Classes

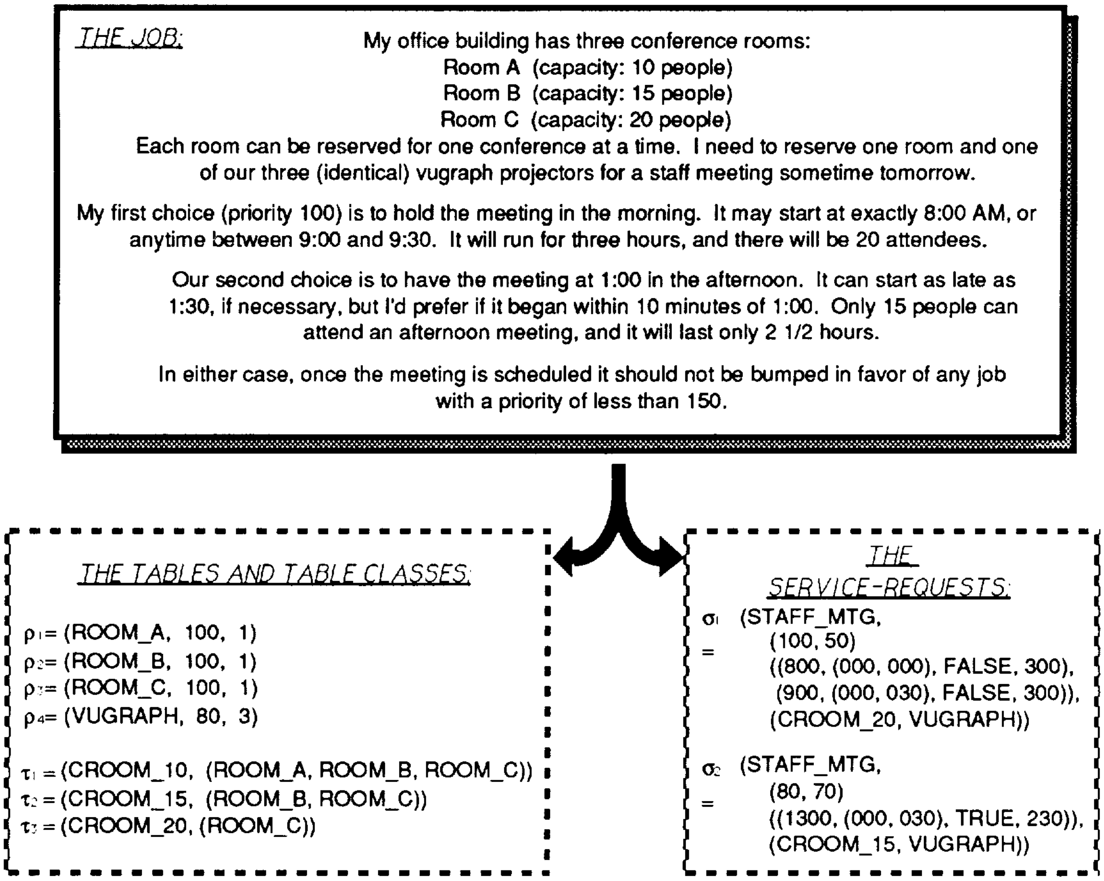
A table is a triplet such as

$$\rho = (\mathbf{r}, \pi_\rho, \lambda)$$
$$P = \{\rho_1, ..., \rho_n\}$$
$$\mathbf{R = \{r(\rho_i) \mid 1 \leq i \leq n\}}$$

where r is the *resource* represented by ρ, π_ρ its *reduction priority*, and λ its *capacity*. Every resource, r_i, has exactly one corresponding table, ρ_i, and vice versa.

The reduction priority, π_ρ, is used during the latter stages of scheduling when SCS assigns a fixed start time and resource set to each job. The higher a table's reduction priority, the more likely its corresponding resource will be used continuously in the final schedule. λ represents the resource's capacity and is specified as a positive integer.

For efficiency, similarly used resources can be collectively expressed as *table classes*. Whenever a service-request specifies a table class, T, $(P \supseteq T)$, in its resource set, SCS will automatically generate N new requests $(N = |T|)$ with the table class replaced by each $\tau \in T$. This allows a user to issue general resource requests such as "one room large enough to hold 20 people" rather than "Either Room B or Room C or Room D or..."



THE JOB:

My office building has three conference rooms:
Room A (capacity: 10 people)
Room B (capacity: 15 people)
Room C (capacity: 20 people)

Each room can be reserved for one conference at a time. I need to reserve one room and one of our three (identical) vugraph projectors for a staff meeting sometime tomorrow.

My first choice (priority 100) is to hold the meeting in the morning. It may start at exactly 8:00 AM, or anytime between 9:00 and 9:30. It will run for three hours, and there will be 20 attendees.

Our second choice is to have the meeting at 1:00 in the afternoon. It can start as late as 1:30, if necessary, but I'd prefer if it began within 10 minutes of 1:00. Only 15 people can attend an afternoon meeting, and it will last only 2 1/2 hours.

In either case, once the meeting is scheduled it should not be bumped in favor of any job with a priority of less than 150.

THE TABLES AND TABLE CLASSES:

ρ_1= (ROOM_A, 100, 1)
ρ_2= (ROOM_B, 100, 1)
ρ_3= (ROOM_C, 100, 1)
ρ_4= (VUGRAPH, 80, 3)

τ_1 = (CROOM_10, (ROOM_A, ROOM_B, ROOM_C))
τ_2 = (CROOM_15, (ROOM_B, ROOM_C))
τ_3 = (CROOM_20, (ROOM_C))

THE SERVICE-REQUESTS:

σ_1 = (STAFF_MTG,
(100, 50)
((800, (000, 000), FALSE, 300),
(900, (000, 030), FALSE, 300)),
(CROOM_20, VUGRAPH))

σ_2 = (STAFF_MTG,
(80, 70)
((1300, (000, 030), TRUE, 230)),
(CROOM_15, VUGRAPH))

Figure 3-1. Service-request and table specifications for the "Staff Meeting" scenario.

An example of how to specify SCS tables and table classes is shown in Figure 3-1. Note that the capacity of each conference room is 1, not 10, 15, or 20. While a room may be large enough to seat up to 20 people, it still can support only one meeting at a time.

3.1.3 Blocks

The output from SCS is a set of blocks representing the mapping of tables to service-requests (i.e., resources to jobs). Blocks are expressed as a 5-tuple:

$$\beta = (\sigma, T_\sigma, P_\sigma, t_w, t_c)$$
$$B = \{\beta_1, ..., \beta_p\}$$

where σ is the service-request associated with β,
T_σ ($\in T(\sigma)$) is the time constraint of σ corresponding to the block
P_σ ($\in P(\sigma)$) is the set of tables in which the block is present
t_w is the *window* of time for the block
t_c is the *critical time* of the block.

The window of a block, t_w, is expressed as an ordered pair (t_{ws}, t_{we}). The length of the window is at least as long as the duration (d) of T_σ. In addition, the window is a subinterval of T_σ's release time and deadline.

The critical time, t_c, specifies the subinterval of t in which the block, if chosen for the final schedule, will definitely be in use. It, too, is expressed as an ordered pair, (t_{cs}, t_{ce}), where $t_{cs} = t_{we} - d$ and $t_{ce} = t_{ws} + d$. If the length of t_w is greater than or equal to two times the block's duration, d, then the block has no critical time, and t_c is expressed simply as "NONE."

To better illustrate critical times, consider a service-request that specifies a release time of 800 hours, a deadline of 1100 hours, and a duration of 200 hours. Such a request could be scheduled from either 800 to 1000, or from 900 to 1100, or from 845 to 1045, etc. No matter which start and end times are chosen, however, the request **must** be in execution between 900 and 1000. Therefore, the block generated from this request would have $t_w(\beta) = (800, 1100)$ and $t_c(\beta) = (900, 1000)$.

The specific types of blocks found in SCS include the following: A *fixed block* is one in which $t_w = t_c$; that is, its window length is exactly equal to its job's duration. A *critical block* is one in which $t_c \neq$ "NONE"; similarly, a *noncritical block* has $t_c =$ "NONE". A *split block* is a special type of noncritical block in which $(t_{we} - t_{ws}) = 2d$. Two blocks are *alternatives* if they share a common job ($\beta_1 \neq \beta_2$ & $j(\beta_1) = j(\beta_2)$), while a *unique* block is one with no alternative.

A generalized definition of schedule can now be given as "any conflict-free set of blocks." A schedule is called *complete* if it consists of only fixed, unique blocks. Schedules that are not complete are *partial* (that is, they contain at least one block which is nonunique and/or nonfixed). Partial schedules are converted to complete ones by assigning fixed start times and resource sets to each job and removing superfluous blocks; this process is called *reduction*. Examples of SCS blocks are given in Figure 3-2.

3.1.4 Notational Conventions

Notational conventions for representing tables and blocks can be defined pictorially. Conflicts, overloaded resources, block alternatives, etc., are much more noticeable when displayed graphically rather than as a textual list of n-tuples.

Figure 3-3 introduces the notational conventions used to represent blocks and tables. The two distinct formats for representing blocks are *standard notation* and *critical notation*. *Standard notation,* which highlights duration and delta time, is best suited for displaying SCS scheduling states. *Critical notation* highlights a block's critical time (or lack thereof) and is useful for identifying conflicts and overloads.

Figure 3-4 shows two blocks, β_1 and β_2, graphed within table ρ_3 ("ROOM_C"), using critical notation.

3.2 SCS Operation

The two operational phases of SCS are the Scheduling Phase and Rescheduling Phase. Its five distinct modes of execution are called Pre-Processing, Placement, Allocation, Completion, and Deallocation. Section 3.2.1 discusses the scheduling strategy used by SCS in each of its various system states.

3.2.1 Scheduling Phase vs. Rescheduling Phase

During the Scheduling Phase, the SCS generates an initial schedule from a static set of service-requests and tables. Then, SCS switches to the Rescheduling Phase in which additions and changes to the initial schedule may be made. Although they are mutually exclusive, both phases share a common set of rules, data structures, computational states, search strategies, and terminology. The most important distinction between the two is that in the Scheduling Phase SCS **creates** a new schedule, while in the Rescheduling Phase SCS **adjusts** an existing schedule.

Scheduling phase is run once, and only once, for any given set of input. After the initial schedule is generated and has been accepted by the user, SCS automatically switches to the Rescheduling Phase. Note that the Scheduling Phase operates on a static set of input data. If the user wishes to make any changes to the input set while SCS is actively generating a schedule, two choices are available: either wait until for Rescheduling Phase and make the changes then, or halt the system, adjust the input, and start the system over.

```
B = { b1 = (σ1, 1,                     b2 = (σ1, 2,
            (ROOM_C, VUGRAPH),                 (ROOM_C, VUGRAPH),
            (800, 1100),                       (900, 1230),
            (800, 1100) )                      (930, 1200) )

      b3 = (σ2, 1,                     b4 = (σ2, 1,
            (ROOM_C, VUGRAPH),                 (ROOM_B, VUGRAPH),
            (1300, 1600),                      (1300, 1600),
            (1330, 1530) )                     (1330, 1530) )  }
```

Figure 3-2. SCS blocks corresponding to the "Staff Meeting" scenario.

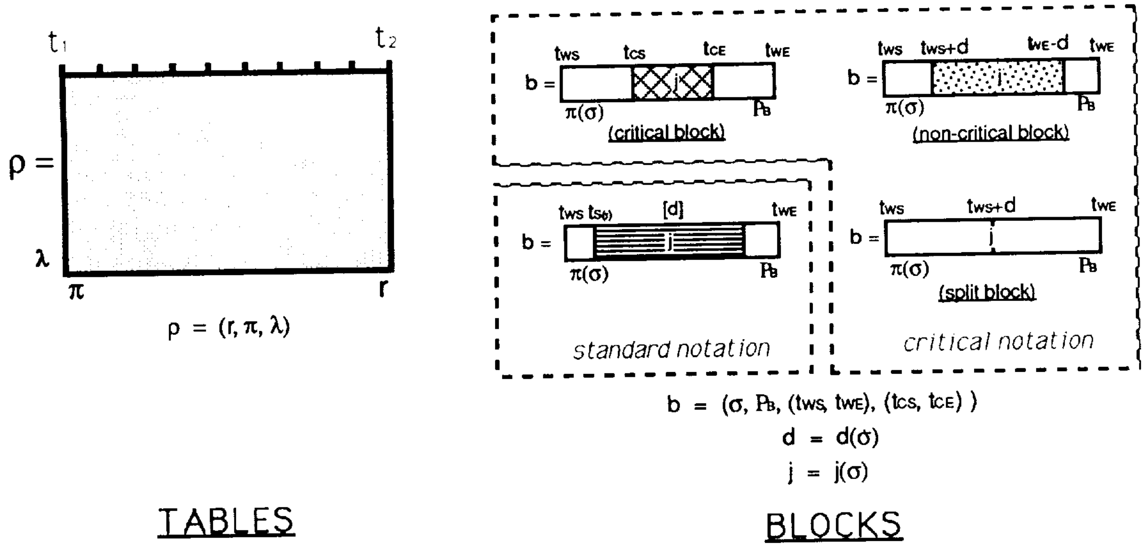


Figure 3-3. Notational conventions.

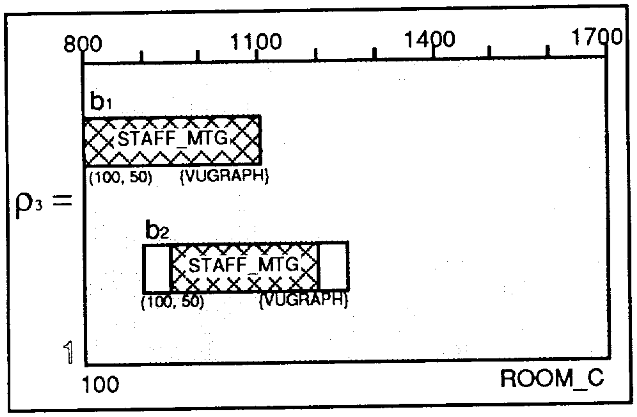


Figure 3-4. Graphing table ROOM_C in the "Staff Meeting" scenarios.

SCS is normally inactive during the Rescheduling Phase, and returns to an active state only when the set of service-requests (Σ) or tables (P) changes. The important feature of the Rescheduling Phase is that it makes *nondisruptive scheduling adjustments*, meaning that it will keep the original schedule as intact as possible during rescheduling. This feature is particularly important for the Space Station because any changes in the C&TS schedule may have a ripple effect on the schedules of other systems (e.g., payload deployment, laboratory experiments, astronauts' personal agendas, etc.).

SCS's projected role for the Space Station is its running in the Scheduling Phase once per evening to generate the C&TS schedule for the next day. SCS remains active in Rescheduling Phase throughout the day to handle last-minute service-requests, device failures, newly activated resources, etc.

3.2.2 Modes of Execution

SCS utilizes a five-step scheduling strategy, with each step known as a *mode of execution*. The SCS system state can be specified as an ordered pair of phase and mode:

$$\begin{aligned} S &= S_p \times S_m \\ &= \{\text{SCHEDULING, RESCHEDULING}\} \times \\ &\quad \{\text{PRE-PROCESSING, PLACEMENT,} \\ &\quad \text{ALLOCATION, COMPLETION,} \\ &\quad \text{DEALLOCATION}\} \end{aligned}$$

If the operational phases (S_p) define the **goal** of SCS (creating a new schedule or adjusting an existing one), then the five modes of execution (S_m) define the **approach** that SCS uses to reach its goal.

SCS's scheduling strategy may be described as a sophisticated generate-and-test algorithm. In summary, one unprocessed service-request is selected and translated into blocks, which in turn are entered in the appropriate tables. Next, SCS analyzes each table to determine when and where conflicts are present. It then uses a collection of algorithmic, heuristic, and blind-search rules to resolve each conflict. Finally, SCS decides whether the new partial schedule is "valid" (i.e., the current request is successfully scheduled, no previously scheduled job has been displaced, and the partial schedule is reducible to some final schedule) or "invalid". In the latter case, SCS restores each modified table to its previous state and marks the current request as "unschedulable." This process is repeated until each job has been processed, at which point SCS fixes a time interval and resource set for each job.

The key to this strategy is that each intermediate partial schedule must be reducible to some final complete schedule such that every job in the former is also in the latter. That is, if you arbitrarily fix any one nonfixed block in the partial schedule, then one method to reduce it to a final schedule must still be available without displacing any jobs. Reduction is defined in more detail in Section 3.2.2.3 on Allocation Mode. Figure 3-5 shows the state transition diagram of SCS.

3.2.2.1 Pre-Processing Mode

Pre-Processing Mode is the first computational state of SCS, the user's input specifications are received and translated to ART™ relations. Unlike the other four execution modes, Pre-Processing Mode is never called explicitly. Rather, it remains in a wait state until new input is received from the user. It then activates, interrupts the current execution mode, processes the new input, and returns to the wait state.

3.2.2.2 Placement Mode

The main computational state of SCS is called Placement Mode. In this mode, service-requests are translated into blocks, and resource conflicts are detected and resolved. Figure 3-6 shows the transition diagram for Placement Mode and its seven substates: Selection, Block Generation, Resolution, Acceptance, Restoration, Rejection, and Displacement.

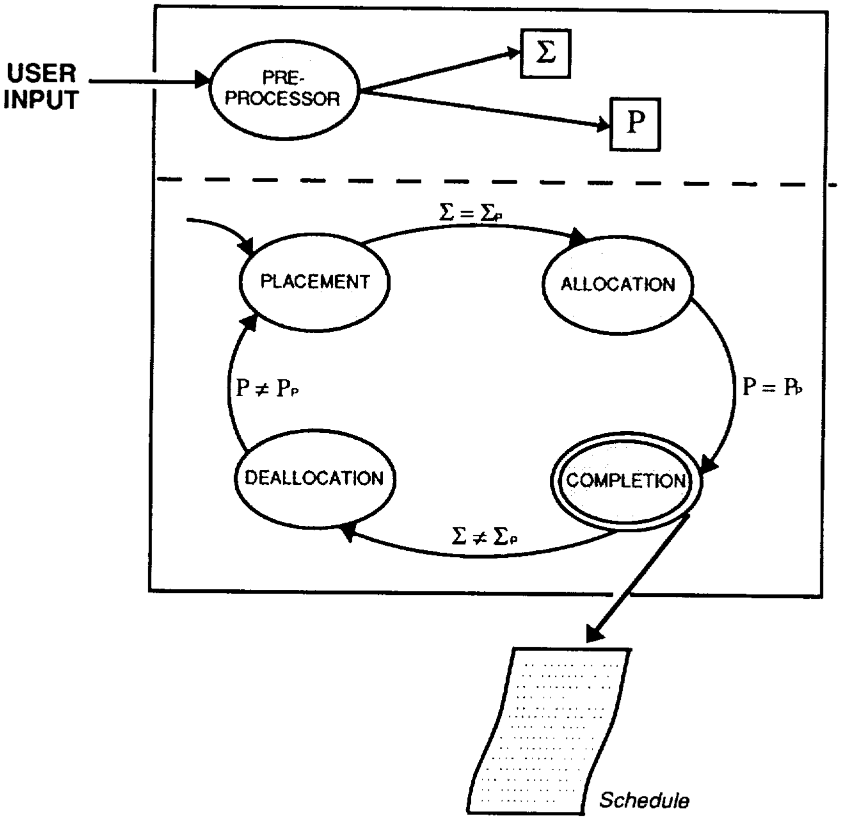


Figure 3-5. State transition diagram for SCS.

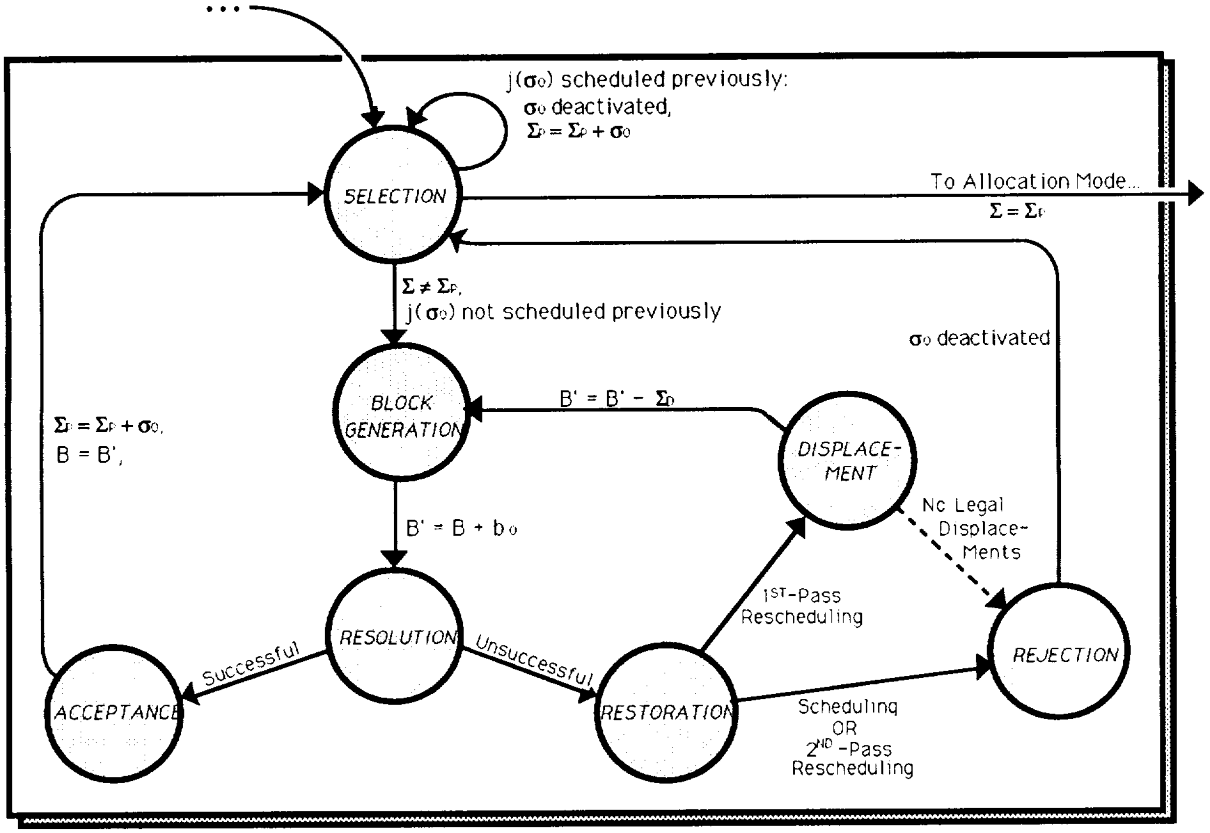


Figure 3-6. Substrate transition diagram for Placement Mode.

SCS operates on one service-request at a time from the *agenda* of unprocessed requests. Recall that the relative scheduling priority, π_p, of each service-request defines a partial order on Σ, and that this partial order determines scheduling order. In the Selection substate, SCS selects the *current-service-request* (represented by σ_0) at random from the set of requests at the top of the agenda. Though using the maximum duration of each request as the second discriminator seems reasonable when selecting σ_0, no improvement using this approach was observed during system testing.

The Block Generation substate performs two functions. First, it checks whether the job corresponding to σ_0 is already present in the partial schedule. Formally, it checks if $\exists\ (\beta \in B)\ (j(\beta) = j(\sigma_0))$. If so, SCS deactivates the request and selects a new σ_0. Otherwise, SCS generates β_0, the set of blocks defined by σ_0, and sets $B' = B \cup \beta_0$. One service-request may generate dozens of alternative blocks for a job, but these will be reduced to, at most, one fixed block in the final schedule.

In the Resolution substate, SCS detects and resolves any conflicts in B′. This substate is by far the most complex component of SCS and is discussed in detail in Section 3.3, "Conflicts and Resolution."

For the current-service-request to be successfully scheduled, B′ must meet two criteria after all conflicts are resolved. First, σ_0 must be represented by at least one block in B′. Second, every job represented by a block in B must also be represented in B′. Formally, $\exists\ (\beta \in B')\ (\sigma(\beta) = \sigma_0) \wedge \forall\ (\beta \in B)\ \exists\ (\beta' \in B')\ (j(\beta') = j(\beta))$. If both criteria are met, then the new partial schedule is accepted.

If either of the two criteria are not met — that is, either σ_0 is not represented in B′, or it "bumped" a job that had been scheduled in B — then the new partial schedule is invalid and the previous partial schedule is restored. The next state transition is dependent on whether SCS is in Scheduling or Rescheduling Phase. In the latter case, the Displacement Substate will attempt to displace blocks from B so that σ_0 can be successfully scheduled (see Section 3.2.3 "Rescheduling Strategy"). If SCS is in Scheduling Phase, or if displacement has already been attempted for σ_0, then the Rejection substate deactivates σ_0 and returns control to the Selection substate.

3.2.2.3 Allocation Mode

When Placement Mode is complete, SCS switches to Allocation Mode. Here, the partial schedule specified by B is reduced to a final, complete one. Each job is assigned a fixed start time and resource set, and its alternative blocks are removed.

Just as Placement Mode processes Σ sequentially according to each request's scheduling priority, Allocation Mode processes P sequentially according to each table's reduction priority. The higher the value of π_ρ for a table, the more likely its corresponding resource will be in continuous use during job execution time.[3] The *current-table* being reduced is denoted ρ_0. Note, however, that the reduction of ρ_0 may cause changes in other tables not yet reduced (because the same block is often present in multiple tables).

The reduction strategy used by Allocation Mode is based loosely on the Fox-Kempf Principle of Opportunism. A huge number of complete schedules may be derivable from one partial schedule, B, thus indicating a heuristic reduction strategy. Consider, though, that certain jobs in J may be represented by only one fixed block (call it $\beta_1 \in B$) in the partial schedule. Because SCS has no option on scheduling this job, and because every effort must be made to keep resources in continuous use, the system should try to find another block, β_2, that can either start when β_1 ends, or end when β_1 starts.

Fixing β_2 next to β_1 creates a *chain* of length two. SCS's reduction strategy is to first extend any existing chains in ρ_0 as long as possible. When no chains are extendible, SCS tries to create a new chain from the remaining set of nonfixed blocks in ρ_0. Only "Minimize-Delta" blocks are exempt from this process; these are fixed as close as possible to their requested start times before any attempt at chaining begins.

When all blocks in B are fixed and unique, Allocation Mode halts and the final schedule is presented to the user. SCS enters Rescheduling Phase (if it is not there already) and waits for new user input in Conclusion Mode.

3.2.3 <u>Rescheduling Strategy</u>

SCS's rescheduling philosophy is to make adjustments to the current schedule with as few disruptions as possible. If a service-request appears on the agenda during Rescheduling Phase — either because the user just issued it, or because one of its allocated resources suddenly became unavailable, or because it was bumped from the final schedule by another job — SCS will first try to schedule it without disturbing any existing blocks. Failing that, SCS will displace certain lower-priority blocks to "squeeze" the new request into the schedule. These bumped jobs are, in turn, placed on the agenda and rescheduled.

Certain jobs will naturally increase in priority once they become part of the final schedule. On the Space Station, for example, the astronauts will arrange their personal schedules according to the daily job schedule they receive each morning. Even a relatively minor job, such as a noncritical scientific experiment, may require extensive preparation time.

The *inertia* of a service-request, $\pi_i(\sigma)$, specifies the difficulty of displacing σ during Rescheduling Phase. Inertia is specified as a nonnegative increment to the request's scheduling priority and σ' cannot bump σ unless $\pi_p(\sigma') > \pi_p(\sigma) + \pi_i(\sigma)$.

The Placement Mode of the Rescheduling Phase will perform two separate attempts to schedule a request on its agenda. During the first pass, this mode operates exactly as in Scheduling Phase unless, and until, the Rejection substate is

[3] The reason why SCS strives to keep resources in continuous use stems from the Space Station domain. To conserve electricity, each device in the Communications and Tracking System is turned off when not in use. All such devices must undergo a "power-up procedure" before being switched back into operation, and this procedure can be relatively expensive in terms of electricity.

reached. Instead of deactivating σ_0, SCS transfers control to a seventh Placement Mode substate called Displacement. Here, SCS creates a *displacement set*, B_D ($B \supseteq B_D$), for σ_0. B_D includes all blocks in B which (1) conflicted with a block $\beta \in B_0$ during first-pass Resolution, and (2) have a rescheduling inertia less than β's scheduling priority. SCS then sets $B = B - B_D$ and returns to the Block Generation substate.

If σ_0 is still unschedulable after the second pass, B_D is wholly restored and control is passed to the Rejection substate as usual. If σ_0 is successfully scheduled, however, then a total restoration of the displacement set will be impossible. Instead, SCS will attempt to restore each block in B_D individually, beginning with the one having the highest inertia. Call the set of unrestorable blocks B_D'. SCS will reactivate all service-requests corresponding to a block in B_D' and place them on the agenda, where they will be rescheduled accordingly.

The Deallocation Mode, also performed during the Rescheduling Phase, is strictly administrative in purpose. In Deallocation, SCS sets P_p, the set of processed tables, equal to the empty set. This ensures that the Allocation Mode, when it is rerun, will process and reduce every table in P.

3.3 Conflicts and Resolution

Conflicts in scheduling and their resolution are an innate part of creating almost any type of schedule. The following example introduces just how SCS addresses such conflicts and resolves them.

Recall the Staff Meeting scenario of Figure 3-1. The first service-request on the agenda, σ_1, requests one 20-seat conference room and one vugraph projector for a 3-hour period, beginning at either 8:00 AM or sometime between 9:00 and 9:30 AM. SCS will generate the two blocks described by this request, then skip over σ_2 because its job is already represented in B.

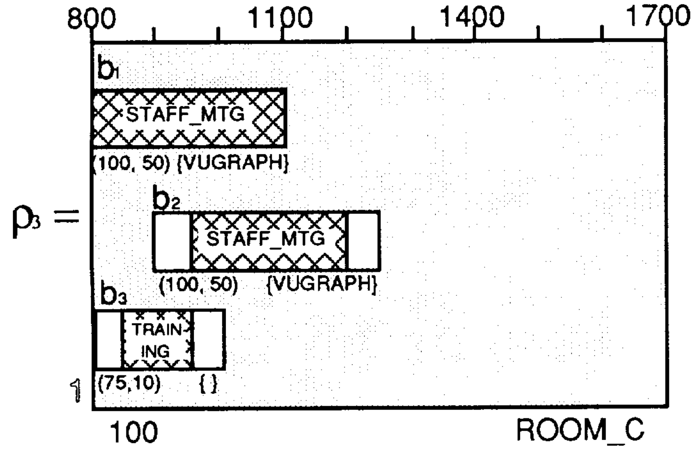
Now assume that σ_3 ($j(\sigma_3)$ = "TRAINING"; $\Pi(\sigma_3)$ = (75, 10)) requests a 20-seat conference room for 1.5 hours starting sometime between 8:00 and 8:30 AM (see Figure 3-7). Only ROOM_C seats 20 people, so this training session will either be held in that room or not held at all. STAFF_MTG has already reserved ROOM_C for most of the morning, but has requested more time than it actually needs. The question is whether a method exists to satisfy the requirements of both jobs.

This is the basis of SCS's *conflict-resolution* scheduling strategy. When the new B_0 is added to B, a number of time intervals will likely have a resource reserved for more jobs than the resource can legally support. The goal of the Resolution substate is to detect and resolve all the conflicts that might prevent B from being reducible to a complete final schedule.

The *Conflict Set* for B is denoted $X = \{\chi_1,\ldots,\chi_n\}$ where each conflict is an ordered triple:

$$\chi_i = (\beta, (t_1,t_2), \Psi).$$

χ_i may be read "block β cannot be scheduled between interval t_1 to t_2 while all blocks in Ψ are



χ_1 = (b2, (900, 930), {b3})
χ_2 = (b1, (830, 930), {b3})
χ_3 = (b3, (800, 1000), {b1})
χ_4 = (b3, (930, 1000), {b2})

χ' = (b3, (930, 1000), {b1, b2})

Figure 3-7. STAFF_MTG vs. TRAINING conflict.

present in B." For every χ_i, the n blocks in $\Psi(\chi_i)$ must represent n distinct jobs.

3.3.1 Conflicts and Decidability

A conflict can be detected as follows: add up the number of distinct jobs present in a table at any time and then check if the sum exceeds the table's capacity. Note, however, that "table" is not one of the parameters in the conflict triplet. Conflicts are a property of blocks alone, and a block may be present in more than one table. In fact, two conflicts found in different tables are often combined to form a third conflict.

Certain types of conflicts are harmless. Figure 3-8a shows a two-block table that can clearly be reduced to a final schedule, despite the presence of an overload at (t_1,t_2). However, the conflict in Figure 3-8b is definitely harmful. One of its two blocks will have to be eliminated if the table is to be reducible. The first type of conflict can be labeled "safe" and the other "dangerous."

A question now arises, does a simple algorithm exist that decides whether any given χ is safe or dangerous? Apparently not. While many conflicts, such as the two in Figure 3-8, are easily decidable, some appear to require nothing short of trial-and-error.

SCS classifies conflicts into three categories, $X = X_S \cup X_D \cup X_U$ (see Figure 3-9). Dangerous conflicts ($\chi \in X_D$) are resolved algorithmically, either by restricting the width of $\beta(\chi)$ or removing it entirely. Similarly, safe conflicts ($\chi \in X_S$) are ignored for the moment, although later changes to the schedule may cause a safe conflict to become dangerous. Any conflict that cannot be easily proven safe or dangerous is called "undecidable" ($\chi \in X_U$).

3.3.2 Dangerous Conflicts

A decision as to the type of conflict is basically a problem of pattern matching. Templates can be defined that describe a certain class of conflict in its simplest form. A pattern matcher would then try to find matches for these templates in a heavily crowded schedule. This type of problem is well suited for a rule-based implementation such as ART™.

Fortunately, the majority of dangerous conflicts fall into three easily defined classes. The most common type is a *first-order conflict.* This conflict occurs in table ρ when the critical times of $\lambda(\rho)$ blocks overlap each other, and these in turn overlap another block. The four conflicts χ_1-χ_4 in Figure 3-7, as well as the one in Figure 3-8a, are first-order conflicts.

A simple *second-order conflict* is shown in Figure 3-10b. Here, noncritical block 4 overlaps the critical times of blocks 1 through 3, but no first-order conflicts are present. Block 5 overlaps block 4 at both points t_A and t_B. Upon examination, it is apparent that block 5 must either (a) start after t_A or (b) end before t_B if block 4 is to be schedulable.

Third-order conflicts (see Figure 3-10c) are closely related to second-order conflicts, except that in this example block 4 is a critical block rather

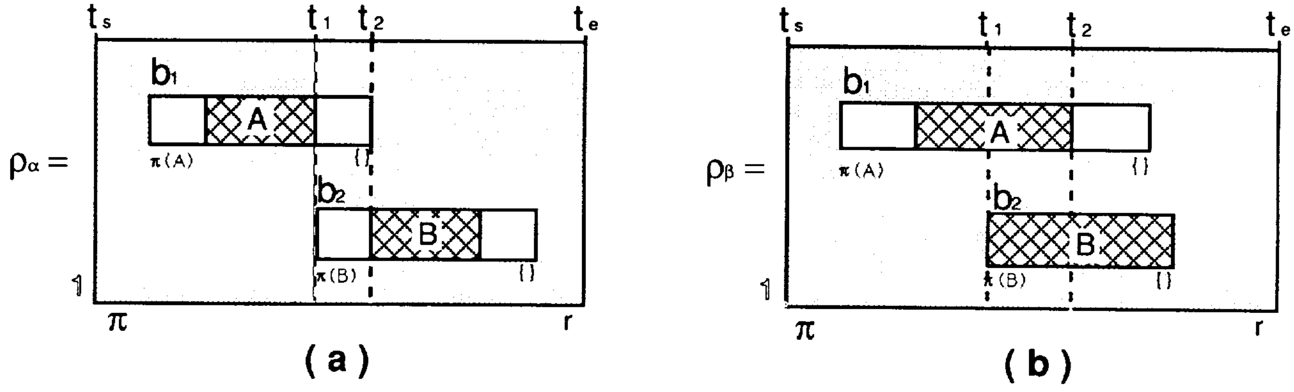


Figure 3-8. Safe vs. dangerous conflicts.

C-2

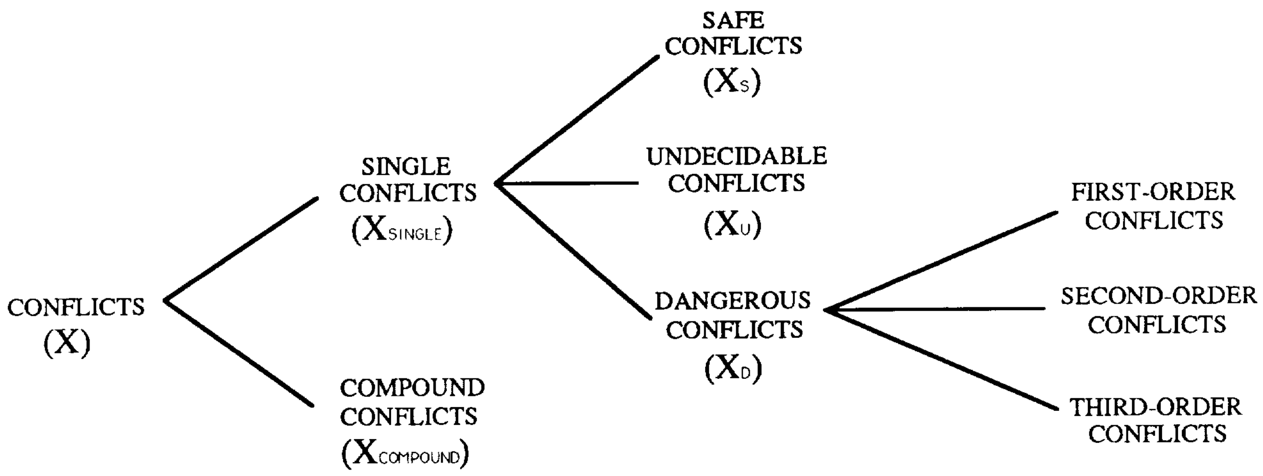
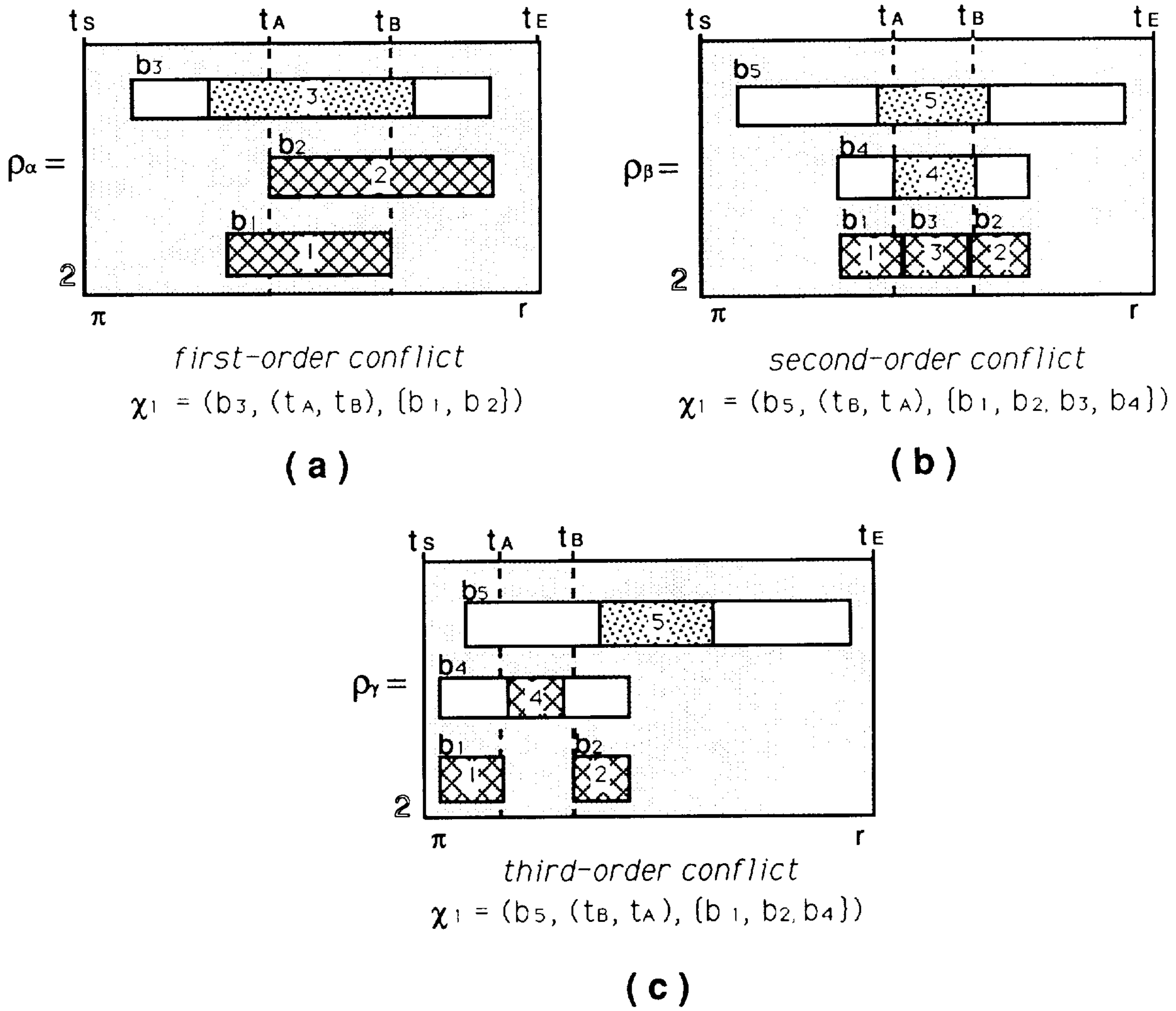


Figure 3-9. SCS conflict classes.



first-order conflict

$\chi_1 = (b_3, (t_A, t_B), \{b_1, b_2\})$

(a)

second-order conflict

$\chi_1 = (b_5, (t_B, t_A), \{b_1, b_2, b_3, b_4\})$

(b)

third-order conflict

$\chi_1 = (b_5, (t_B, t_A), \{b_1, b_2, b_4\})$

(c)

Figure 3-10. Dangerous conflict classes recognized by SCS.

than a noncritical one. Again, note that no first-order conflicts are present and that block 5 must start after t_A if block 4 is to be scheduled.

Both second- and third-order conflicts have $t_2(\chi)$ less than $t_1(\chi)$. It seems strange to say that a conflict is present "between 10 AM and 9 AM". However, if one considers $t_1(\chi)$ to be the latest safe time that $\beta(\chi)$ can end, and $t_2(\chi)$ to be the earliest safe time that $\beta(\chi)$ can start, then this order of specifying times is consistent for all three conflict classes. Note also that the job duration of $\beta(\chi)$ must be greater than $(t_1(\chi) - t_2(\chi))$ minutes for any conflict to be valid.

3.3.3 Operations on Conflicts

As described previously, the presence of a dangerous conflict simply means that a certain block cannot be scheduled concurrently with certain other blocks. How such a conflict should be resolved, or even if it should be resolved, is not always clear.

Consider Figure 3-11. This table clearly contains a dangerous first-order conflict, χ = (2, (t_1, t_2), {1}). Does this mean that block 2 must be restricted to start after t_2? Not necessarily. If blocks 1 and 1A are alternatives, that is, j(1) = j(1A), two options are available: block 2 can be restricted, or block 1 can be removed. Which option is "correct" depends on the service-requests yet to be processed.

A dangerous conflict with more than one possible resolution is called an *open* conflict. The Fox-Kempf Principle of Least Commitment calls for the decision on resolving open conflicts to be delayed as long as possible. A *closed* conflict is one which has only one possible resolution, in which case SCS can make the necessary adjustment immediately.

Now consider Figure 3-12 which has two dangerous first-order conflicts: χ_1 and χ_2. Assume j(1A) = j(1B). Both conflicts are open when examined separately, but notice that if block 2 is scheduled across time t_2, then neither block 1A nor 1B is schedulable. A new closed conflict has appeared from the intersection of two open ones: χ' = (2, (t_2, t_2), {1A, 1B}).

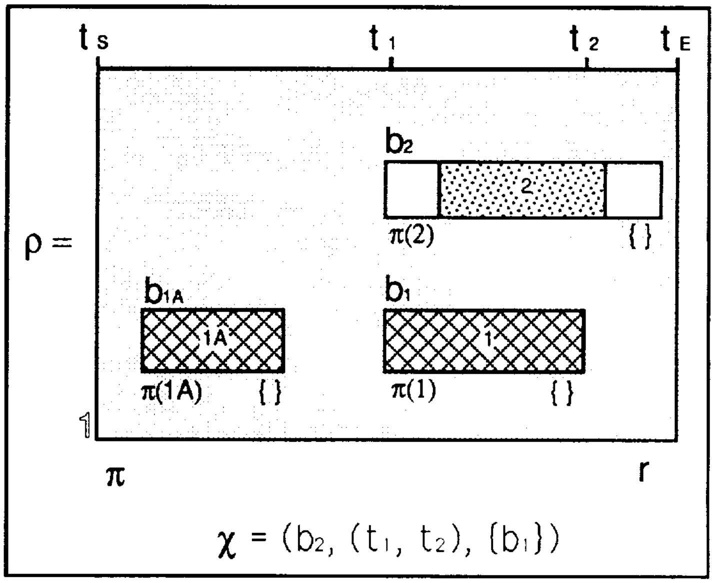


Figure 3-11. An "open" dangerous conflict.

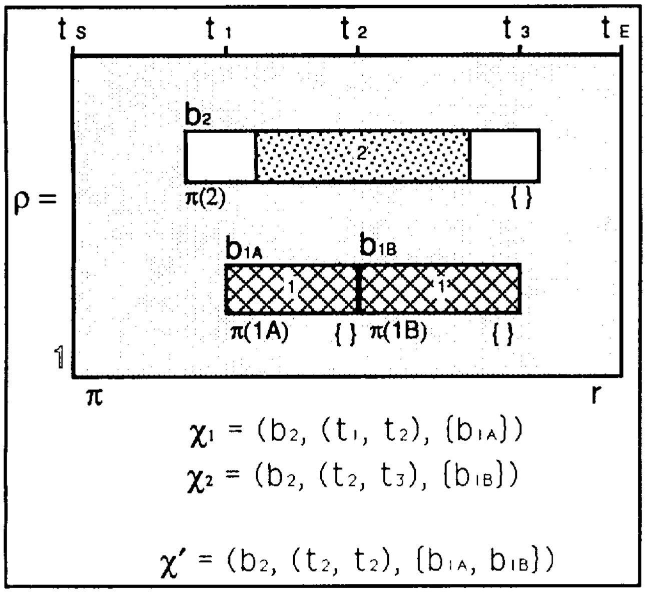
Combining two open conflicts in this manner yields a *compound conflict* (represented χ'). Compound conflicts are always dangerous, though they may be open or closed. They differ from single conflicts in that two or more blocks in $\Psi(\chi')$ may represent the same job. A complex set of rules governs when and how two conflicts may be combined.

The criteria for determining whether any conflict, single or compound, is open or closed can now be addressed. Given χ = (β, (t_1, t_2), Ψ), if no job in $\Psi(\chi)$ has an alternative not contained in $\Psi(\chi)$, then χ is closed.

SCS resolves closed conflicts by moving the offending block completely out of the conflict interval. Specifically, for any closed conflict χ = (β, (t_1, t_2), Ψ), SCS will try to split β into two new blocks: one running from $t_{ws}(\beta)$ to $t_1(\chi)$, the other from $t_2(\chi)$ to $t_{we}(\beta)$. Of course, if a new block is not wide enough to support j(β), then it is removed from B.

3.3.4 Undecidable Conflicts and Resolution States

As stated earlier, SCS is unable to make decisons concerning certain dangerous conflict classes. Most of these occur in tables having an overabundance of noncritical blocks. Figure 3-13 illustrates one such example. None of the three



t s
t 1
t 2
t 3
t E
ρ =
b2
2
π(2)
{ }
b1A
b1B
1
1
π(1A)
{ } π(1B)
{ }
1
π
r
χ1 = (b2, (t1, t2), {b1A})
χ2 = (b2, (t2, t3), {b1B})
χ′ = (b2, (t2, t2), {b1A, b1B})

Figure 3-12. A "closed" compound conflict.

known conflict orders are present, but no method exists to reduce this table without removing one of the 13 blocks.

Although a scheduler should ideally not make firm scheduling decisions until absolutely necessary, SCS requires that all partial schedules be reducible at the conclusion of each Resolution substate. SCS must, therefore, assume that all undecidable conflicts are dangerous. At this point, SCS can still heuristically adjust the schedule to minimize the chance that a later service-request will be precluded unnecessarily. However, SCS's first priority is to schedule the current service-request, and it might become necessary to make some restrictive decisions to squeeze σ_0 into B.

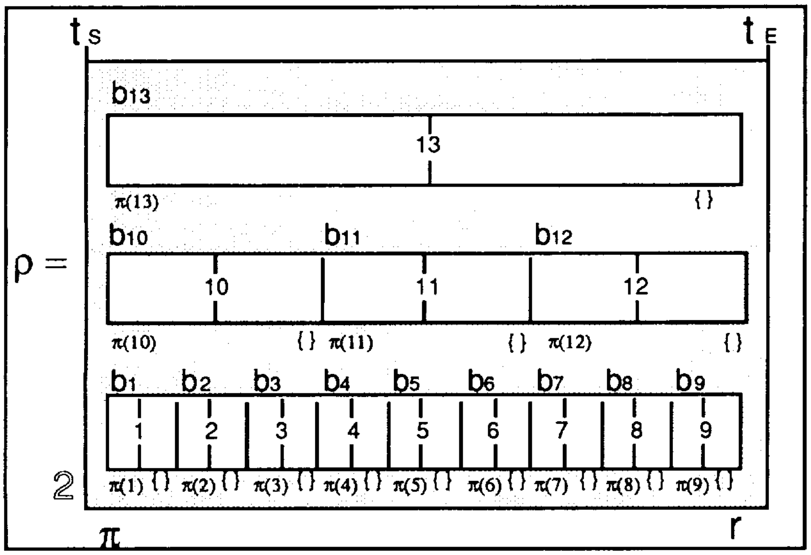


Figure 3-13. An undetectable dangerous conflict.

The obvious first step is to do nothing about undecidable conflicts until all closed conflicts have been detected and resolved. Resolving one closed conflict often greatly decreases $|X|$ because it may eliminate a block that was part of many other conflicts.

Step two is to safely eliminate an undecidable conflict, either by removing the overload condition that is causing it or by converting it into a safe one. Simply nudging a block out of a conflict interval or eliminating it completely, if it has an alternative that is not part of any conflict, often eliminates an undecidable conflict. SCS uses a set of heuristics to choose an effective, relatively nondisruptive adjustment, then checks if any existing conflicts may be combined or closed as a result.

If these methods fail, step three is to do whatever is required to successfully schedule σ_0, no matter what effect this may have on the scheduling of future requests. SCS checks the most promising search paths looking for any successful and reducible partial schedule.

The longer SCS requires to eliminate X_U, the less flexibility SCS has to deal with later service-requests. Consequently, the more conflict classes that are decidable, the better quality schedule SCS will produce. However, the execution time of SCS is directly proportional to the number of decidable conflict classes.

In the Staff Meeting example discussed at the beginning of this section, SCS recognizes four dangerous conflicts when σ_3's blocks are added to B (χ_1 to χ_4; see Figure 3-7). A fifth, compound conflict (χ_1') is generated by merging χ_3 and χ_4. Three of these are closable (χ_1, χ_2, χ_1'), and the resulting partial schedule is shown in Figure 3-14.

4.0 IMPLEMENTATION

SCS has been implemented on a Digital VAX-8650 mainframe under the VMS operating system. SCS contains over 300 ART production rules, and 1000 lines of custom C and LISP code. The user supplies an ASCII file containing definitions of the service-requests, tables, and scheduling parameters.

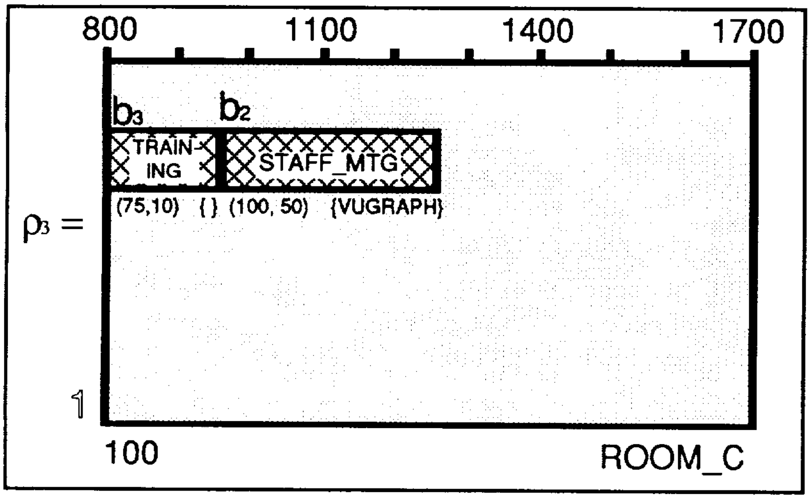
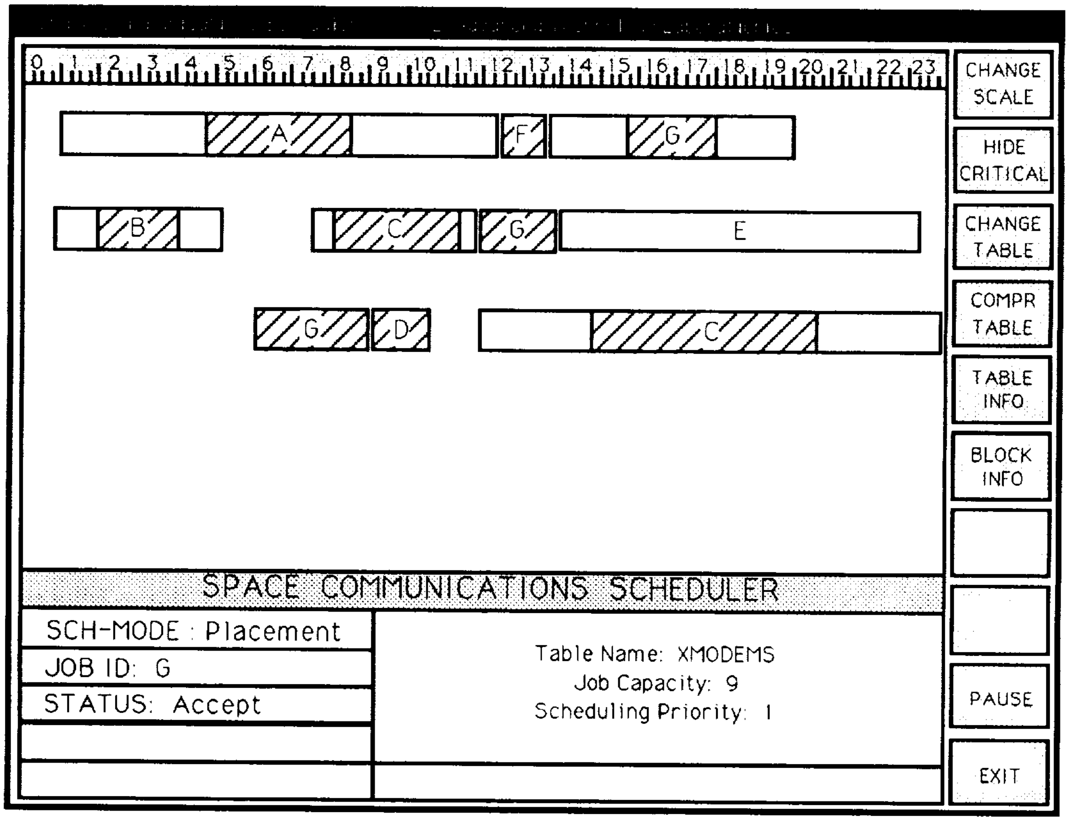


Figure 3-14. The "Staff Meeting" scenario of Figure 3-1 with all conflicts resolved.

A graphical user interface to SCS has also been developed using a Digital VT341 color terminal (see Figure 4-1). Tables and blocks are represented in graphical format, with the user having the option of displaying or suppressing critical times. Mousing on a corresponding graphic obtains information on tables or blocks. A column of mouse icons along the right edge of the screen allows the user to enter commands to SCS.

The user interface may be run as a coprocess or parent process to SCS. The scheduler generates a series of one-line ASCII messages that notifies the interface program when a significant action has been performed.



SCH-MODE : Placement
JOB ID: G
STATUS: Accept

Table Name: XMODEMS
Job Capacity: 9
Scheduling Priority: 1

Figure 4-1. SCS user interface.

These actions include creation of a table, creation of a new block, adjustment or deletion of an existing block, selection of a new current-service-request, or a change in the system's phase or mode. The user interface acts on these messages sequentially, adjusting the display to reflect the new system state.

5.0 CONCLUSIONS AND FUTURE WORK

The conflict-resolution scheduling strategy of SCS works quite well within SCS's limited scheduling subclass. Empirical data indicates that SCS operates in low-order polynomial time for $|P|$ (the number of tables) and $|J|$ (the number of distinct jobs). It appears, however, to be exponential for MAX_λ, the maximum capacity of any $\rho \in P$ (hence the requirement for low-capacity resources).

Future development work may address variable-length job durations ("I need a conference room for between three and four hours"), non-reusable resources, and resource sets with nonidentical time constraints ("I need a conference room for two hours, and a vugraph projector for the first half-hour"). Another useful feature would be to allow inertia values to be specified as a function of time, based on the theory that it is better to reschedule a job ten hours before its scheduled start time rather than just ten minutes beforehand. SCS may also be translated into C or Ada to improve speed and facilitate software verification. Before any translation can occur, however, an efficient means for detecting dangerous conflicts is needed because SCS will no longer have access to ART^{TM}'s powerful pattern-matching facility.

SCS could also be extended to allow temporal restrictions between jobs. For example, a specification could be made that job j_1 may not begin executing until j_2 halts. Restrictions could also be specified at the service-request level ("if j_1 is scheduled via σ_1, then σ_2 must run concurrently with it"), or they could define one job to be contingent on another ("If j_1 is scheduled before 0800 hours, then do not schedule j_2"). Allen [9] lists 13 possible relationships between time intervals that could be used as the bases for temporal restrictions.

REFERENCES

Garey, M.R., and D.S. Johnson; *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York, NY: W.H. Freeman and Co., 1979.

Martin, C.F., and R.S. Poling; "Fast, Dynamic Programming Selection Algorithm for the Job-Shop Problem with Job Availability Intervals", GE Technical Information Series, No. 78CIS012, 1978.

Marcus, R.; "An Application of Artificial Intelligence to Operations Research", *Communications of the ACM*, Vol. 27, No. 10; pp. 1044-1047; October1984.

Deitel, H.M.; *An Introduction to Operating Systems.* Reading, MA: Addison-Wesley, 1984.

Tanenbaum, A.S.; *Operating Systems - Design and Implementation.* Englewood Cliffs, NJ: Prentice-Hall Inc.,1987.

Nillson, N.J.; *Principles of Artificial Intelligence.* Palo Alto, CA: Tioga Publishing Co., 1980.

Hayes-Roth, B., F. Hayes-Roth, S. Rosenschein, and S. Cammarata; "Modeling Planning as an Incremental, Opportunistic Process", *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*; IJCAI-79, Tokyo, pp. 375-383, 1979.

Fox, B.R., and K.G. Kempf; "Complexity, Uncertainty and Opportunistic Scheduling", *Proceedings of the Second Conference on Artificial Intelligence Applications. Washington, DC*: IEEE Computer Society Press, pp. 487-492, 1985.

Allen, J.F.; "Maintaining Knowledge About Temporal Intervals", *Communicatons of the ACM*, Vol. 26, No. 11; pp. 832-843, November 1983.

Constraint-Based Evaluation of Sequential Procedures

Matthew R. Barry

Rockwell Space Operations Company
NASA/Johnson Space Center DF63
Houston, TX 77058
mbarry@nasamail.nasa.gov

19 January 1990

Abstract

Constraining the operation of an agent requires knowledge of the restrictions to physical and temporal capabilities of that agent, as well as an inherent understanding of the desires being processed by that agent. Usually a set of constraints are available that must be adhered to in order to foster safe operations. In the worst case, violation of a constraint may be cause to terminate operation. If the agent is carrying out a plan, then a method for predicting the agent's desires, and therefore possible constraint violations, is required. The conceptualization of *constraint-based reasoning* used herein assumes that a system knows how to select a constraint for application as well as how to apply that constraint once it is selected.

The purpose of this paper is to discuss the application of constraint-based reasoning for evaluating certain kinds of plans known as *sequential procedures*. By decomposing these plans, it is possible to apply context- dependent constraints in production system fashion without incorporating knowledge of the original planning process.

As an illustration of these ideas, this paper presents a system used in the Space Shuttle Mission Control Center to evaluate propulsive consumables management plans.

1 Introduction

We being with the assumption that a planning system has defined a plan to achieve some (desirable) goal. Normally, the planning agent passes the plan to an executor agent in order to achieve the goal. If the planner and the executor are independent, however, there may be situations in which misinterpretations or invalid instructions occur. These situations may occur if the processes assume different operation states, run asynchronously, or are modified frequently.

Rather than discovering a problem when an improper action is attempted, the output of the planning agent can be evaluated by an intermediate agent using a set of constraints governing the operation of the executor agent. If the evaluation process uncovers conflicts between the actions specified in a plan and the actions executable by the intended agent, then execution of that plan is inhibited until a mediator resolves the conflict. Though these constraints normally would be considered in an automated planner's line of reasoning, it might be the case that a plan is generated manually. In this case the evaluation process acts as an assistant to the human plan developer.

2 Plans

A plan specifies a means for accomplishing a goal[1]. The collection of actions defining the plan might be unique for each world in which the plan is applied. Furthermore, the results of applying the plan are dependent upon

[1] This paper does not investigate the various techniques for constructing plans (see instead [Steel 1987,Wilkins 1988,Stefik 1981] or the volume edited by Georgeff and Lansky [Georgeff and Lansky 1987]).

the initial state before any of the constituent actions are undertaken. We can sometimes conceptualize the overall plan as consisting of a set of shorter plans, with each element of the set possibly operating in a unique context.

We select a universe of discourse and a set of constraints that specify how the actions occurring in the changing world are to be applied in the current state. We define a *state* as a representation of the current situation in the world. An *action* changes the state of the world. An *action block* consists of a finite sequence of actions. A *conditional action* consists of a satisfaction condition and two different actions. If the condition evaluates *true* in the current state, then one of the two actions is selected. Otherwise, the second action is selected.

A *sequential procedure* maps positive integers into the action that is to be performed at the corresponding step of an infinite sequence [Genesereth 1987]. For example, a sequential procedure P enumerates the order of application for some actions $ActionI$ and $ActionJ$:

$$\begin{aligned} P(1) &= ActionI(x) \\ P(2) &= ActionJ(x) \\ P(3) &= ActionJ(y) \end{aligned}$$

where $ActionI(x)$ denotes the application of the object constant $ActionI$ to the object x, and so on. The state of the world at the end of the sequential procedure is the result of applying each action in turn beginning with some initial state for which the plan was generated.

2.1 Assumptions

In order to suitably restrict the kind of plans we can reason about, we make the following assumptions:

1. The agent assigned to carry out the actions will assume that the plan is executable and satisfies the goal.[2]

[2] In some cases there may be multiple agents available to operate in parallel, each on a different part of the plan [Lansky 1987]. Plans for these situations may require

2. The overall plan is decomposable into a finite sequence of smaller plans.

3. There are no conditional actions.

4. None of the action blocks overlap (the plan is linear).

3 Constraints

Since we assume that another agent created the plan, we must validate that agent's work. To do this we check that the plan satisfies the operational constraints of the executor agent. The constraints considered herein evaluate both the structure and content of a plan.

3.1 Identification

Three sorts of constraints are defined for suitably-restricted plans: *internal*, *local*, and *global*. *Internal* constraints apply to the semantic content of action or action block objects. These constraints validate the object itself, rather than its existential purpose.

Local constraints apply to the event currently under consideration as well as the events occurring just before it. These constraints are independent of the plan context. They represent physical system limitations, temporal requirements, and operational management techniques. Usually we can reason about *local* constraints as *action blocks*.

Global constraints apply to all of the actions in the plan, and are dependent upon the evaluation context.

an *interagent* constraint evaluation among differing contexts. Pednault [Pednault 1987] describes a technique for reducing some plans intended for multiple agents into a plan for a single agent. Such a plan may introduce a contextual evaluation based on *boundary conditions*.

3.2 Application

All of these constraints manifest themselves as *production rules* in the plan evaluation system. Each rule represents one constraint. Certain groupings of rules permit preprocessing and postprocessing activities, which might be context dependent. All of the contexts encountered during the evaluation are maintained in the *context memory*, which essentially is a database of running sequences, accomplished events, unaccomplished events, etc.

The implementation described below uses standard *production rules* to represent the constraints. This is convenient due to the nature of most constraints. Typically they read “Only do step B after step A is complete,” or “Shutdown if value V of component X exceeds threshold Y.” These statements might be captured with production rules like

```
if not Complete(Step(A))
then Pause(Step(b)).
```

or

```
if X.V > Y
then Shutdown.
```

4 Example

4.1 Background

One of the duties assigned to the Propulsion team in the Mission Control Center (MCC) is to maintain a propellant budgetting plan for all scheduled activities through the end of the mission. These plans allot propellant to future maneuvers and attitude maintenance activities. Furthermore, the team must ensure that certain minimum propellant quantities, or “redlines”, are available at various points in the sequence. These plans are constructed initially before launch, but are updated frequently during the course of the mission. The plan must always be an accurate representation of the activities to be carried out by the astronauts. Various constraints dictate proper implementation of maneuver sequences, redline construction, mandatory activities, etc. Violation of the redlines is cause to terminate a mission abruptly.

A propellant budget usually consists of a few hundred records itemizing each of the maneuvers and attitude maintenance periods. This implies the special case of *sequential procedures* within the previously defined plan restrictions. Each record in the plan represents an action to be performed, and the *immediate* context applies only to that record. Certain of the rules apply to the *immediate* context constraints. Some of these rules are especially important for verifying the validity of propellant usage references. That is, they verify that the propellant cost for a particular action is (1) non-zero, and (2) the proper budget item for the *global* context.

Most of the rules apply to *local* constraints. The *local* context consists of the current record and the few records before and after it in the plan, or an *action block*. *Local* constraints limit action durations (e.g. a maneuver should not last more than 20 minutes), adjacent actions (consecutive OMS burns are not realistic), and action modes (no primary FRCS thrusters firing during crew sleep periods).

The evaluation process levies constraints against the entire plan as well as to each action comprising the plan. Within the every plan there must appear certain actions, and these actions must appear in a certain order, regardless of how many actions separate them. The *global* context can be derived from the name of the plan file or from the first actions appearing in the plan (the evaluator assumes these represent the persistent context). The *global* context determines which data files are to be accessed, which constraint limits are to be applied, etc.

4.2 Implementation

The example system was coded in **awk** running on a UNIX workstation. The *pattern / action* constructs processed by **awk** represent the production rules. Some machinery was built around these constructs to manage the context memory, to control iteration, and to manage data files. Though **awk** runs as an interpreter, the full application evaluates a plan consisting of several hundred actions in only a few seconds. This level of performance is quite acceptable considering the utility of the output and the potential time saved in manually debugging a plan.

The evaluator only displays problem conditions: it does not fix the problem itself.[3] A typical problem list may look like the following:

```
Vernier timeline evaluation:
(1) ERROR: Differing attitudes without maneuver (line 130).
(2) ERROR: Invalid event time (line 151).
(3) WARNING: RCS Hotfire occurs before FCS Checkout.
Processed 260 lines.
```

Here the integer reference to the *sequential procedure* step number sometimes appears in the problem context description. The first **ERROR** message above might have been due to the sequence

$$\begin{aligned} P(129) &= AttHold(180,0,270) \\ P(130) &= AttHold(270,0,270) \end{aligned}$$

whereas a correct implementation of the (virtual) action block might be

$$\begin{aligned} P(129) &= AttHold(180,0,270) \\ P(130) &= Maneuver(90,0,0) \\ P(131) &= AttHold(270,0,270) \end{aligned}$$

The example application uses the declarative programming paradigm to distinct advantage. The constraints involved in plan evaluation typically are ill-ordered, being applicable whenever the constrained situation arises, not as a sequential application of other constraints. By applying constraints

[3] Though it certainly could do so for errors occurring in an unambiguous context.

through a production system, the application is able to accommodate additional constraints without regard to the computational sequence. Procedural techniques which accomplish the same sort of reasoning are certainly possible, however the declarative techniques are easier to implement.[4] An equivalent system might also be coded in CLIPS, LISP or some other readily available substitute.

5 Conclusions

A simple technique for evaluating sequential procedures by applying operational constraints has been presented. This technique is useful for determining the feasibility of carrying out a plan that was created without rigorous knowledge of the constraints imposed by the executor agent.

The evaluation process incorporated into the Space Shuttle consumables planning programs strives to eliminate the mistakes commonly made when developing propellant budgets. This process provides real-time quality assurance for these critical products. It uncovers subtle problems that might go unnoticed until further downstream in the development effort by applying constraints to various aspects of the plan. It provides context sensitive reasoning capabilities that the (human) plan developers might overlook. Most importantly, it is flexible to enhancement, easily accommodating constraint modifications.

References

[Genesereth 1987] Genesereth and Nilsson, *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1987.

[Georgeff and Lansky 1987] Georgeff and Lansky (eds.), *Reasoning About*

[4] Moreover, they better represent the actual evaluation process carried out by experts.

Actions and Plans: Proceedings of the 1986 Workshop, Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1987.

[Lansky 1987] Lansky, "A Representation of Parallel Activity Based on Events, Structure and Causality," in *Reasoning About Actions and Plans: Proceedings of the 1986 Workshop*, Georgeff and Lansky (eds.), Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1987.

[Pednault 1987] Pednault, "Formulating Multiagent, Dynamic-World Problems in the Classical Planning Framework," in *Reasoning About Actions and Plans: Proceedings of the 1986 Workshop*, Georgeff and Lansky (eds.), Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1987.

[Steel 1987] Steel, "Topics in Planning," in *Advanced Topics in Artificial Intelligence*, Nossum (ed.), Springer-Verlag, Berlin, 1987.

[Stefik 1981] Stefik, "Planning and Meta-Planning (MOLGEN: Part 2)," *Artificial Intelligence 16*, 1981.

[Wilkins 1988] Wilkins, *Practical Planning: Extending the Classical AI Planning Paradigm*, Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1988.

N90-22302

SURE (Science User Resource Expert): A Science Planning and Scheduling Assistant for a Resource Based Environment

Nancy E. Thalman Thomas P. Sparn
Laboratory for Atmospheric and Space Physics
University of Colorado at Boulder
Campus Box 10
Boulder, Colorado 80309

Abstract

SURE (Science User Resource Expert) is one of three components that compose the SURPASS (Science User Resource Planning and Scheduling System). This system is a planning and scheduling tool which supports distributed planning and scheduling, based on resource allocation and optimization. The SURPASS written in Ada, uses a DEC windows user interface (X based), the INGRES database management system and the SURE system. The SURE system uses the CLIPS/Ada expert system production shell. SURPASS is designed to support a wide range of science applications and can be easily tailored via database modifications, DEC window updates and rule specifications for the SURE system.

Currently SURE is being used within the SURPASS by the UARS (Upper Atmospheric Research Satellite) SOLSTICE instrument to build a daily science plan and activity schedule and in a prototyping effort with NASA GSFC to demonstrate distributed planning and scheduling for the SOLSTICE II instrument on the Eos platform.

For the SOLSTICE application the SURE utilizes a rule-based system. Development of a rule-based program using Ada CLIPS as opposed to using conventional programing, allows for capture of the science planning and scheduling heuristics in rules and provides flexibility in inserting or removing rules as the scientific objectives and mission constraints change. An additional advantage of rule-based programming is that it facilitates the representation of the relationship between instrument operations and resources. SURE uses these rule sets to implicitly assist the science user in planning within the context of the science goals while optimizing instrument operations using the available resources.

This paper describes the SURE system's role as a component in the SURPASS, the purpose of the SURE planning and scheduling tool, the SURE knowledge base, and the software architecture of the SURE component.

This is a synopsis of the paper which was unavailable at the time of publishing

Introduction

The Science Users Resource Expert, SURE is a resource-oriented, knowledge-based, planning and scheduling tool component designed for use by the science planner and instrument scheduler. The SURE system separates the Science Planning Context from the Resource Scheduling Context by allowing the user to plan and schedule instrument activities with respect to scientific goals while maximizing instrument activity with respect to available resources.

The SURE system is currently being developed by the Laboratory for Atmospheric and Space Physics (LASP) at the University of Colorado in Boulder. The SURE system is used within the SURPASS(Science User Resource Planning and Scheduling System) by the SOLSTICE instrument to plan and schedule science experiments and instrument activities. The SOLSTICE will fly on the UARS(Upper Atmospheric Research Satellite) in 1991 and Eos (Earth Observing System) in 1997. SURPASS is being developed jointly under the SCAN (Scheduling Concepts Architectures and Networks) study sponsored by GSFC code 522 and the UARS SOLSTICE project. The SURE component is being developed under the SCAN testbed study.

The SURPASS system, as illustrated in Figure 1, is composed of three Ada language-based software components, one component being the SURE system, and the other two components, the User Interface(UI) and the Planning and Scheduling System(PASS) Manager. The User Interface is written using the X-based DEC windows package and contains timeline and application specific displays that display the planning and scheduling information within the science context. The PASS Manager component is responsible for data handling, communication, and transactions between the three SURPASS components. The SURE component is written using the CLIPS/Ada expert system production shell. The SURE system incorporates a rule base captured from the knowledge of the planning expert, the instrument engineers and the scientist. The rule base is intended to optimize resource utilization based on the desired science objectives.

CLIPS/Ada is a forward chaining rule language based on the Rete algorithm and is written in Ada and developed by the NASA Johnson Space Center. Using the CLIPS/Ada expert system production shell has allowed

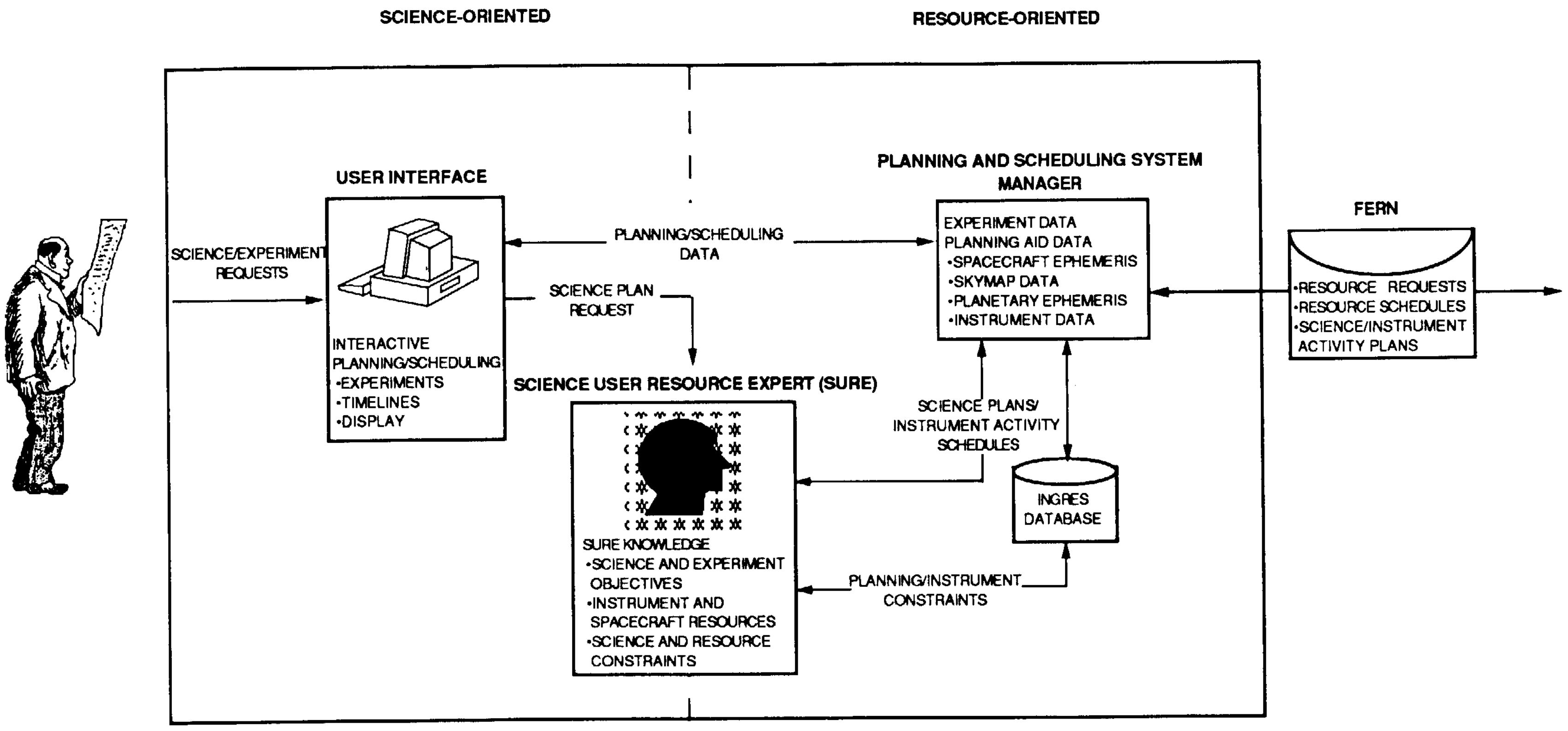

SCIENCE-ORIENTED
RESOURCE-ORIENTED
SCIENCE/EXPERIMENT REQUESTS
USER INTERFACE
INTERACTIVE PLANNING/SCHEDULING
•EXPERIMENTS
•TIMELINES
•DISPLAY
PLANNING/SCHEDULING DATA
SCIENCE PLAN REQUEST
SCIENCE USER RESOURCE EXPERT (SURE)
SURE KNOWLEDGE
•SCIENCE AND EXPERIMENT OBJECTIVES
•INSTRUMENT AND SPACECRAFT RESOURCES
•SCIENCE AND RESOURCE CONSTRAINTS
PLANNING AND SCHEDULING SYSTEM MANAGER
EXPERIMENT DATA
PLANNING AID DATA
•SPACECRAFT EPHEMERIS
•SKYMAP DATA
•PLANETARY EPHEMERIS
•INSTRUMENT DATA
SCIENCE PLANS/ INSTRUMENT ACTIVITY SCHEDULES
INGRES DATABASE
PLANNING/INSTRUMENT CONSTRAINTS
FERN
•RESOURCE REQUESTS
•RESOURCE SCHEDULES
•SCIENCE/INSTRUMENT ACTIVITY PLANS

Figure 1 SCIENCE USER RESOURCE PLANNING AND SCHEDULING SYSTEM

rapid prototyping of an instrument planning and scheduling knowledge base. Since the User Interface and PASS Manager components are also written in Ada, this has allowed for easy interfacing and integration of the SURPASS components.

The SURE system's goal is to produce a schedule of science activities and observations within a dynamic resource environment. These observations and activities are scheduled in a way that maximizes instrument activity and hence, the scientific results within the available resource envelopes. Using an expert system has allowed the user to plan science experiments which satisfy science observing objectives and still remain within the given resource constraints without having to schedule these activities based solely on resource availability. The SURE system also reschedules instrument activities based on changing or updated resource availability while maintaining the overall scientific objectives.

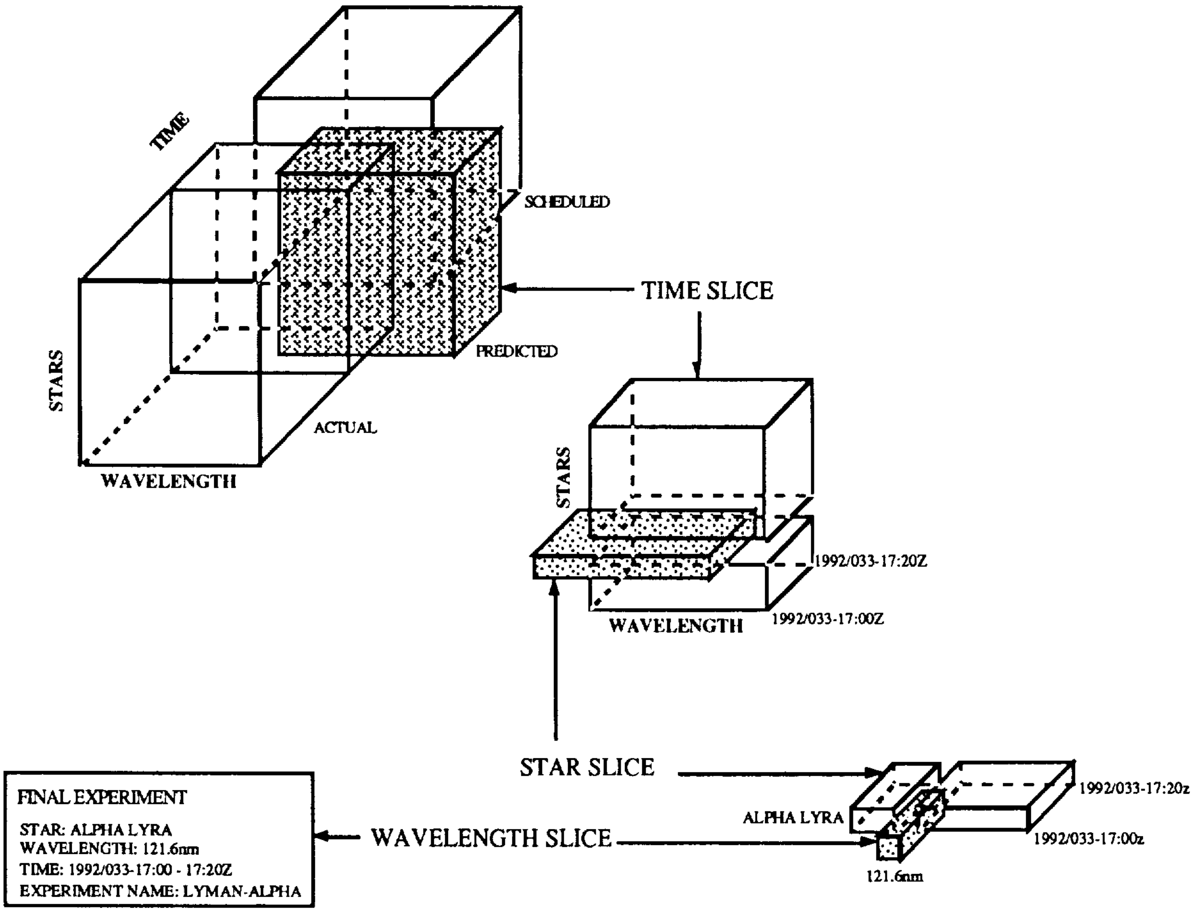
Additional features of the SURE system include its ability to constraint check manually generated science plans. SURE will calculate resource requirements, check resource availability constraints, and attempts to reschedule an activity or request additional resources if needed. SURE notifies the user if the activity cannot be inserted into the schedule due to resource or constraint conflicts and may suggest alternate activities or actions.

The SURE Knowledge Base

Development of a knowledge-based system as opposed to using a conventional programming methodology, allows for the representation and capture of knowledge by the scientist, the instrument operator and planner, and the instrument engineers into rules. These rules combine to provide the translation of scientific goals, instrument operating modes and resource requirements, and operational constraints into a optimized resource based schedule and instrument sequence. These rules also allow flexibility as instrument operating characteristics and the instrument environment change requiring different resources levels.

For this application, rules have been designed to select specific solar observation times from an initial preallocation of available solar resources and based on scientific goals, determine the solar experiments to be executed during the selected solar observation times. These experiments are selected to fit within available resource envelopes. If an experiment can not be found based on the scientific goals, increased resources may be requested. The next step is to determine the stellar viewing periods and for each period select specific stars to be observed based on star observation statistics.

Star observation statistics which reside in the data volume are the actual, scheduled, and planned experiment data maintained on each star. The observations statistics from the data volume guide the expert system's selection of star experiments. The goal is to equally observe each star and it's corresponding wavelengths, so that the data volume is uniformly dense throughout. The data volume concept is illustrated by Figure 2.



FINAL EXPERIMENT

STAR: ALPHA LYRA
WAVELENGTH: 121.6nm
TIME: 1992/033-17:00 - 17:20Z
EXPERIMENT NAME: LYMAN-ALPHA

Figure 2 DATA VOLUME CONCEPT

For the selected stars an optimal instrument tracking plan to minimize instrument dead time within operating resource envelopes is then determined. Once the maximum stellar dwell time is determined for each target, experiments are selected based on the scientific goals. If experiments do not fit with the determined time additional resources may be requested to increase the dwell time. The resulting schedule is an initial science plan with a optimal target acquisition sequence which contains scheduled solar and stellar experiments. This schedule is passed to the PASS manager and User Interface for the user to review.

The schedule produced by the SURE system is based on broad scientific goals and may not reflect detailed short term changes to scientific objectives. The user may interact with the initial schedule generated by the SURE system through the User Interface to include enhanced science objectives. SURE aids the user interaction by constraint checking user inputs with respect to available resources.

SURE Software Architecture

The SURE software architecture, as illustrated in Figure 3, consists of a SURE task, several Ada packages, and the CLIPS/Ada expert system production shell. The SURE task is an Ada task containing a rendezvous for starting up the expert system. For the UARS SOLSTICE application this rendezvous is with the PASS Manager component of SURPASS. The SURE_INTERFACE_PKG contains procedures which are responsible for obtaining resource availability data and resource constraints from the STAR_INTERVALS_PKG. The SURE_INTERFACE_PKG procedures take the resource availability and constraint data and build facts for the expert system, load the SURE rule set into the CLIPS/Ada inference engine and start the execution of the expert system. Internally, the inference engine uses facts, available resources and star observation statistics, and tries to match these facts with rules from the knowledge base (see Figure 4). In the UARS SOLSTICE case, the expert system appropriately generates either scheduled solar and stellar experiments, scheduled slew times, or resource constraint error messages. These are transmitted from the SURE task via Ada rendezvous with the PASS Manager. These scheduled experiments are made available in working memory to the User Interface which displays the experiments on a timeline or in application specific windows.

The SURE system uses the CLIPS/Ada production shell which contains a forward-chaining inference engine and the SURE rule set has been written using the CLIPS/Ada rule-based language. The SURE rule set

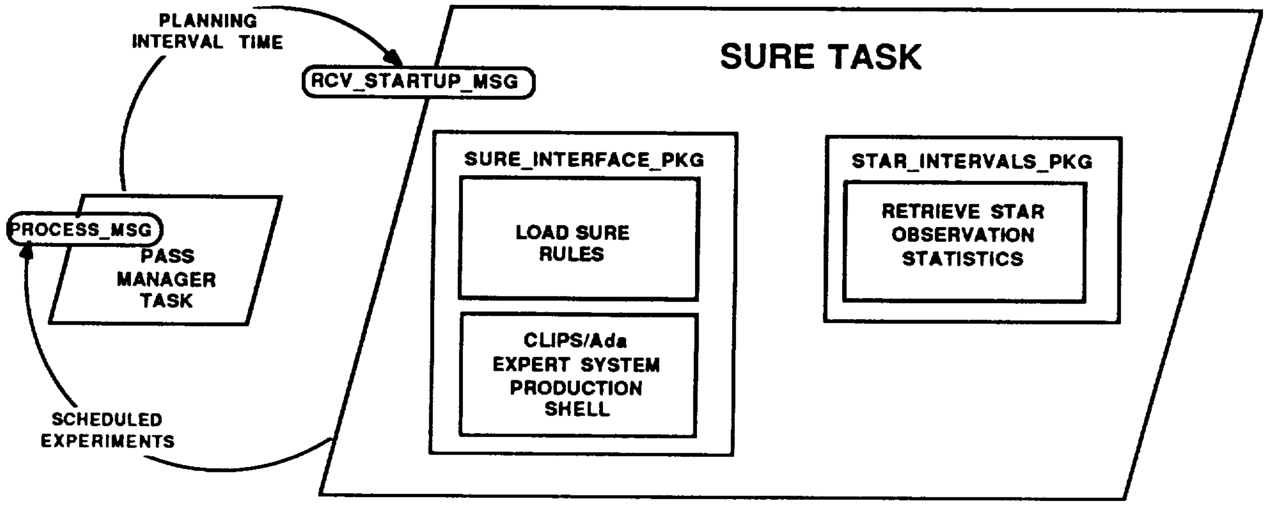


Figure 3 SCIENCE USERS RESOURCE EXPERT SOFTWARE ARCHITECTURE

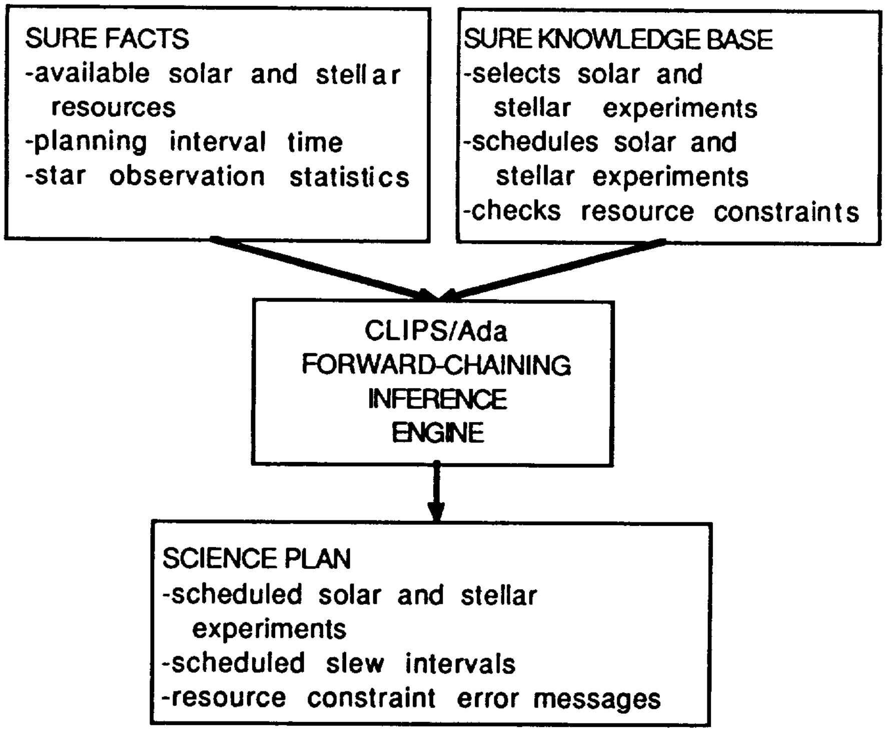

SURE FACTS
-available solar and stellar resources
-planning interval time
-star observation statistics
SURE KNOWLEDGE BASE
-selects solar and stellar experiments
-schedules solar and stellar experiments
-checks resource constraints
CLIPS/Ada FORWARD-CHAINING INFERENCE ENGINE
SCIENCE PLAN
-scheduled solar and stellar experiments
-scheduled slew intervals
-resource constraint error messages

Figure 4 SOLSTICE EXPERT SYSTEM

is designed to select available resources needed for experiments and schedule resources in a manner which maximizes their usage. The SURE rule set or knowledge base can be modified and tailored for other applications by the development of a different rule base.

As a planning/scheduling tool the expert system allows the user to spend more time concentrating on scientific goals and less time scheduling experimental activities and available resources. For the UARS SOLSTICE application, the human planner/scheduler takes approximately 6 hours to produce a 24 hour science plan. The SURE system using rules reduces the time to schedule a 24 hour day to only 30 minutes. This is a significant reduction in scheduling time.

Conclusions

Using an expert system such as the SURE knowledge-based application frees the user from becoming a planning or scheduling expert and allows the science user to plan and schedule instrument activities with respect to the project's scientific goals. The SURE expert planner/scheduler builds an experiment plan that is based upon scientific goals and takes care of the details of instrument operations and resources. The SURE expert system is an application independent scheduler and can be tailored and adapted for use with other instrument control systems.

Acknowledgements

The authors would like to thank and acknowledge the following persons for their suggestions and assistance in the preparation of the paper: Jeanette Fielden, Dan Gablehouse, David Judd, and Kristin Stordahl.

References

Giarratano, J. (August 1989). Clips User's Guide. Artificial Intelligence Section, Lyndon B. Johnson Space Center, Houston, Texas.

Hansen, E. R., Sparn, T. P., and Davis, R. L. (May 1988). Concepts for Planning and Scheduling in the Space Station Era. Laboratory for Atmospheric and Space Physics, University of Colorado, Boulder, Colorado.

Hansen, E. R., and Sparn, T. P. (1989). Concepts in Distributed Scheduling and Control. Conference on Space Station Evolution: Beyond the Baseline. League City, Texas.

Hull, L. (October 1989). Space Station A/D Program Trip Report. Goddard Space Flight Center, Greenbelt, Maryland.

Melebeck, C. J. (November 1989). Clips/Ada Advanced Programming Guide. Barrios Technology, Houston, Texas.

A KNOWLEDGE-BASED APPROACH TO IMPROVING OPTIMIZATION TECHNIQUES IN SYSTEM PLANNING

J. A. Momoh Z. Z. Zhang

Dept. of Electrical Engineering
Howard University
Washington, DC 20059
(Tel: (202) 636-5454/5350)

ABSTRACT

The paper presents a knowledge-based (KB) approach to improve mathematical programming techniques used in the system planning environment. The KB system assists in selecting appropriate optimization algorithms, objective functions, constraints and parameters. The scheme is implemented by integrating symbolic computation of rules derived from operator and planner's experience and is used for generalized optimization packages.

The KB optimization software package is capable of improving the overall planning process which includes correction of given violations. The method has been demonstrated on a large-scale power system discussed in the paper.

Keywords: System Optimization, System Planning, Expert System, Power System, Security Analysis, Optimal Power Flow.

INTRODUCTION

The planning of large-scale system requires the use of optimization techniques and software programs. Many of the optimization algorithms developed to solve planning problems include recursive quadratic programming, the cost function method, the feasible direction method, etc. All of these methods have certain common calculation during each iteration. They all need appropriate selection of objective functions, constraints, prioritization of contingencies and some mechanism to enforce global convergence.

The implementation of these algorithms calls for human judgment to efficiently use the software package dedicated for optimization tools available. To achieve an improvement over traditionally used planning processes, the technique uses a staged approach to planning and employs a knowledge-based support to assure optimal performance of available optimization packages.

The paper is organized into four major sections. Section one deals with the concept of knowledge-based hierarchical optimization (KBHO). Section two deals with application of the knowledge-based system to security assessment in planning. The third section implements the expert system for power systems problem,while section four gives a summary of results and concluding remarks.

THE CONCEPT OF KNOWLEDGE-BASED HIERARCHICAL OPTIMIZATION

The knowledge-based hierarchical optimization (KBHO) is conceived for improving system-wide planning problems for a typical large-scale system. The generalized optimization problem is formulated below. The stages of planning process and the rules of the expert system are identified.

Formulation of Optimization Problem

The optimization problem of large-scale systems can be formulated as given by (Toint, edited by Osiabacz, Clarendon Press, 1988).

$$\min\ f(x) \qquad (1)$$

subject to

$$l \leq X \leq u, \quad (2)$$

$$AX \leq b, \quad (3)$$

$$CX \leq 0, \quad (4)$$

where f: $R^n \rightarrow R$ is the objective function, l, u $\in R^n$ are the lower and upper bounds on the variables, A is an m x n matrix, b $\in R^m$ and C: $R^n \rightarrow R^m$ represents the non-linear constraints.

Stages of Planning Process

Three different stages are included in the planning process. Figure [1] displays the basic structure and the interactions between the various stages.

(a) System Modeling

A modeling description of the planning problem is given by the planner. This involves the selection of available models and preparation of input data. KB support is provided at this stage to improve the data and to select the appropriate optimization model.

(b) System Planning

A schedule of the plan needed to achieve a specified scenario is given, while at the same time the validity of the plan is checked. If the plan fails, or is unavailable, we return to stage (a) to modify or change the model description.

(c) System Simulation

The selected plan and optimization model is simulated until the planner is satisfied with the generated plan. If the plan turns out to be infeasible, the proposed KB checks the algorithm for convergence and suggests remedial action to improve the plan or adjust the parameters.

The solution of the optimization problem stated in equations (1) through (4) is implemented to guarantee an optimum solution by optimizing the stages discussed above. The proposed expert system implements the planning process by performing staged sequential performance of decision-making process. It couples knowledge-based components with numerical computation programs. By using a hierarchically structured data transfer, storage and updating of data and knowledge is improved. Thus, the overall planning scheme is enhanced. The four major levels of the suggested KBHO scheme is described below.

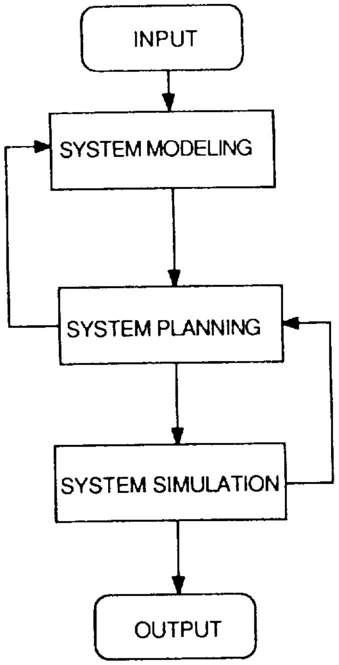


FIGURE 1: A BASIC PLANNING PROCESS

Level I: Identification of Hierarchical and Multi-level Decision-Making Process

This level identifies the numerical computation process at a decision-making (DM) subprocess. The options considered are appropriate modeling, algorithms needed for given tasks, selection of constraints and objective functions, evaluation of parameters and convergence criteria.

Level II: Knowledge-Based and Optimal Selection of the Options

This level employs the KB support to select the options constructed in the upper level. This is accomplished by using deductive reasoning and flexible man-machine interaction at lower levels. The selection is based on fully integrated information and available optimization techniques.

Level III: Implementation of Mathematical Programming Problem

This level solves the proposed optimization problem by using the appropriate mathematical programming method. Several methods are available and selection according to task is done by employing knowledge-based support in IV.

Level IV: Implementation of KB Synthesis

This level employs the expert system at the top level of the planning process. It guarantees a system-wide optimum by ensuring staged optimization. It assists in the coupling of numeric and symbolic computation program modules, and also guarantees man-machine interaction between the user and the KB planning process.

EXAMPLE OF SYSTEM PLANNING PROBLEM

Th optimal power flow (OPF) has been successfully employed in the electric power industry to determine the optimum allocation and scheduling of power systems. Optimal power flow has also been used for security assessment due to the impact of loss of line or unit contingency. (Alsac et al., IEEE, 1974) have developed an off-line optimization scheme based on nonlinear optimization to determine specified objectives and constraints.

The proposed KBHO is designed for on-line use and is capable of selecting different objective functions and associated constraints. The evaluation of the impact of contingencies and the selection of optimum strategy have been improved.

Optimal Power Flow Problem

In general, the optimal power flow problem in normal operation of power system is described mathematically as follows:

$$\min F(x, u) \qquad (5)$$

subject to

$$g(x, u, p) = 0 \qquad (6)$$

$$h(x, u, p) \leq 0 \qquad (7)$$

where X is the state variable vector for voltage magnitudes and angles, u is the controllable variable vector for generation outputs and transformer taps, etc and P is the uncontrollable variable vector for admittances in networks. The function F(x, u) is the objective function representing operation costs, power loss, voltage deviation, etc. Equation (6) denotes equality constraints and equation (7) represents inequality constraints.

Options of Objective Functions and Constraints

Several options of objective functions and associated constraints are developed in quadratic form as shown in [Momoh, SMC 1989].

The objective function is given as

$$F(x) = 1/2X^T RX + a^T X \qquad (8)$$

for loss, cost, voltage deviation and their combinations. Their associated constraints are defined as

$$F_i(x) = 1/2X^T H_i X + b^T X = K_i \qquad (9)$$

where $K_{i1} \leq K_i \leq K_{i2}$; $i = 1, 2, \ldots m$.
and K_{i2} and K_{i1} are upper and lower limits constraints.

Details of the objective functions and constraints are given in Tables 1 and 2. Various parameters and variables of the objective functions and the constraints are defined in Tables 3, 4 and 5. Table 6 gives a summary of the various possible combinations of the constraints for each objective function.

Objective Functions	FORMULATION	DESCRIPTIONS
OF_1	$\frac{1}{2}x^T Dx - DV^T x$	Voltage Deviation Minimization
OF_2	$\frac{1}{2}P^T_{KE} R_C P + a^T_C P_{KE}$	Cost Minimization
OF_3	$\frac{1}{2}P^T_{KE} R_L P_{KE} + a^T_L P_{KE}$	Loss Minimization
OF_4	$W_1 OF_2 + W_2 OF_3$	Production Cost & Loss Min.
OF_5	$W_1 OF_1 + W_2 OF_2 + W_2 OF_3$	Voltage + Current + Loss

TABLE 1: SUMMARY OF DESCRIPTIONS OF OBJECTIVE FUNCTIONS

Constraints	FORMULATION	DESCRIPTIONS
Const. 1	$V^2_{KE\min} \le \lvert V_{KE}\rvert^2 = \frac{1}{2}x^T H_v x + b_v^T x \le V^2_{KE\max}$	Voltage Constraints
Const. 2	$P_{KE\min} \le P_{KE} = \frac{1}{2}x^T H_{2k-1} x + b_p^T x \le P_{KE\max}$	Real Power Constraints for Generator
Const. 3	$Q_{KE\min} \le Q_{KE} = \frac{1}{2}x^T H_{2k} x + b_g^T x \le Q_{KE\max}$	Reactive Power Constraints for Generator
Const. 4	$P_{KD\min} \le P_{KD} = \frac{1}{2}x^T H_{2k+1} x + b_D^T x \le P_{KD\max}$	Real Load Power Constraints
Const. 5	$P_{DE\min} \le P_{DE} = \sum_{i}^{g} (1-B_i)P_i - \sum_{i=1}^{g}\sum_{j=1}^{g} P_i B_{ij} P_j \le P_{DE\max}$	Demand Constraints
Const. 6	$\lvert I_{ij}\rvert^2_{\min} \le \lvert I_{ij}\rvert = \frac{1}{2}x^T H_I x \le \lvert I_{ij}\rvert_{\max}$	Current Constraints

TABLE 2: Summary of Constraints

Variables	DESCRIPTION
x	Voltage vector, 2n x 1, real and imaginary part of voltage
V	Expected voltage value vector, 2n x 1
P_{KE}	Real Power of Generator, g x 1.
h	Bus number of of a given system.
g	Generator number of a given system.

TABLE 3: Objective Function Variable Description

Variables	DESCRIPTION
D	Weight Coeffieciency Matrix, diagonal matrix with positive elements
R_C	$R_c = \mathrm{Diag}[\theta_1, \theta_2, \cdots, \theta_g]$ θ_i is the cost efficiency of the generator
a_C	$a_C = [b_1, b_2, \cdots, b_g]$, b_i is the cost efficiency of the generator
R_L	$R_L = \{ B_{ij} \}$ is the matrix of Loss coefficiency wjth $B_{ji} = B$
a_L	$a_L = [B_1, B_2, \cdots, B_g]$, B_i is the Loss coefficiency

TABLE 4: Objective Function Parameter Description

Variables	DESCRIPTION
H_v	All elements equal to zero bu$(2k-1)^{th}$ and $2K^{th}$ elements equal to 2
H_I	Has at most eight non-zero elements $H_I(2i-1, 2k-1) = H(2j-1, 2j-1) = H_I(2i, 2i) = H_I(2j, 2j) = 2(G^2_{ij} + B^2_{ij})$ $H_I(2i-1, 2j-1) = H_I(2j-1, 2i-1) = H_I(2i, 2j) = H_I(2i, 2j) = -2(G^2_{ij} + B^2_{ij})$
H_{2k-1}	$\begin{bmatrix} 0 & \begin{matrix} B_{k1} & G_{k1} \\ -G_{k1} & B_{k1} \end{matrix} & 0 \\ & \vdots & \\ B_{k1} \;\; -G_{k1} \cdots & 2B_{kk} \;\; 0 & \cdots B_{kn} \;\; -G_{kn} \\ G_{k1} \;\; -B_{k1} \cdots & 0 \;\; 2B_{kn} & \cdots -B_{kn} \;\; G_{kn} \\ & \vdots & \\ 0 & \begin{matrix} B_{kn} & G_{kn} \\ -G_{kn} & B_{kn} \end{matrix} & 0 \end{bmatrix}$
H_{2k}	$\begin{bmatrix} 0 & \begin{matrix} G_{k1} & -B_{k1} \\ B_{k1} & G_{k1} \end{matrix} & 0 \\ & \vdots & \\ G_{k1} \;\; B_{k1} \cdots & 2G_{kk} \;\; 0 & \cdots G_{kn} \;\; B_{kn} \\ -B_{k1} \;\; G_{k1} \cdots & 0 \;\; 2G_{kn} & \cdots -B_{kn} \;\; G_{kn} \\ & \vdots & \\ 0 & \begin{matrix} G_{kn} & -B_{kn} \\ B_{kn} & G_{kn} \end{matrix} & 0 \end{bmatrix}$
b_v, b_p, b_g, b_D	Linear term coefficiency, depend on the system
B_i B_{ij}	See objective function parameter description See objective function parameter description

TABLE 5: Constraint Parameter Description

Objective Function	Constraints			
OF_1	Constraint 1	Constraint 2	Constraint 3	Constraint 4
OF_2	Constraint 1	Constraint 2	Constraint 3	Constraint 5
OF_3	Constraint 1	Constraint 2	Constraint 3	Constraint 5
OF_4	Constraint 1	Constraint 2	Constraint 4	Constraint 6
OF_5	Constraint 1	Constraint 2	Constraint 3	Constraint 5

TABLE 6: SUMMARY OF OBJECTIVE FUNCTIONS AND CONSTRAINTS

Statement of Static Security Assessment Problem

The static security assessment (SSA) problem is concerned with answering the folowing question: How should the power system be operated so that failures do not cause problems? In answering this question successfully, it is important that operators know which equipment outages will cause flows or voltages to fall outside limits so that they can take appropriate measures in dealing with the harmful outages in order to maintain the system operation within safe limits.

The OPF-based static security assessment problem can be described as the performance of the following staged tasks (Thomas, EPRI, 1988) in Figure 2:

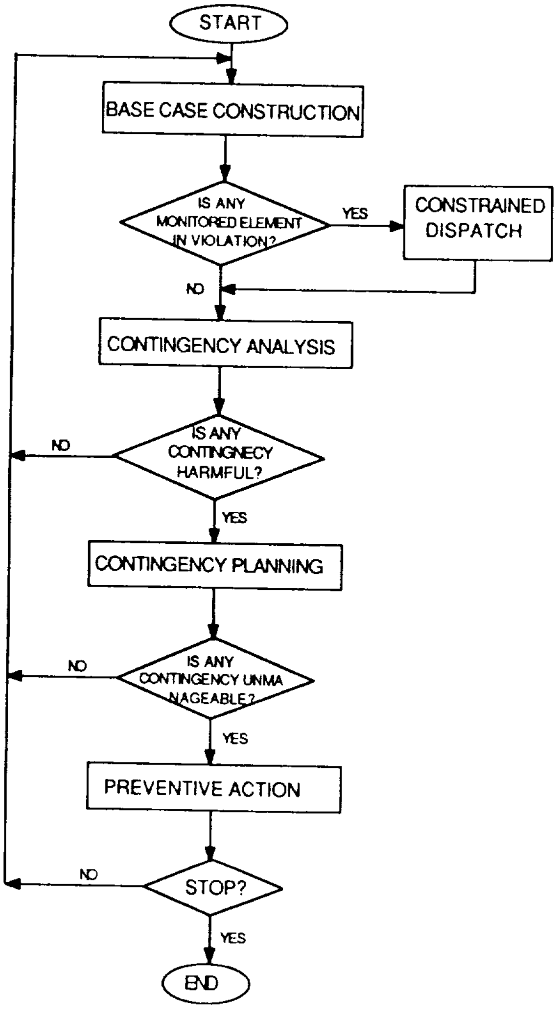


FIGURE 2: A FLOWCHART OF SSA PROCESS

1. Base Case Construction

The goal of this module is to obtain the base case of the power flows and bus voltages in the existing system configuration for further analysis.

2. Constrained Dispatch

The goal of this module is to call the OPF-based corrective scheme of power system operation to correct the violations from the base case.

3. Contingency Analysis

The goal of this module is to determine harmful contingencies for further planning. It also eliminates harmless contingencies from further consideration

4. Contingency Planning

The goal of this module is to determine the corrective scheme that would correct the harmful contingencies. This involves solving the OPF problem. The corrective scheme is saved for operator call up should the contingency occur.

5. Preventive Action

The goal of this module is to determine a preventive correction scheme if one or more contingencies are found to be unmanageable. It also involves solving the OPF problem.

KNOWLEDGE-BASED IMPLEMENTATION

To improve the SSA scheme shown in Figure 2, a knowledge-based approach descussed earlier is employed. The knowledge base selects which constraints and objective functions are appropriate to correct given violations. It employs KBHO methodology to characterize the planning process and combines the various subtasks by coupling numeric computation and symbolic computation. The expert system scheme is built as described in the next section of the paper.

Expert System Design

The expert system is designed to support on-line planning of the OPF-based SSA. The design consists of several knowledge bases dedicated to improving the overall algorithms used in the SSA. Some of the areas of potential improvement are discussed as follows.

1. Partition system condition of the objective power system to effectively reduce the number of contingencies to be studied.

2. Categorize contingencies into critical and noncritical types.

3. Determine appropriate weights selection to represent given violation.

4. Select performance index to best match the operational status of power system.

5. Identify masking and misranking phenomena which characterize performance index used by (Ejebe, IEEE, 1979).

6. Select objective functions and constraints which will be appropriate for a given option in the planning and execution of SSA process.

The expert system is designed in four languages (DCL, C, OPS83, and FORTRAN) and includes three procedure modules as shown in Figure 3. Module 1 consists of numerical programs which are written in FORTRAN for both the Automatic Contingency Selection (ACS) and the optimization process. The second module, which performs symbolic computation, consists of symbolic programs where the rules developed are coded in OPS83. The representative rules describing each of the knowledge bases are constructed by using a forward chaining mechanism. The third module displays the interface program modules written in DCL and C and enhances man-machine.

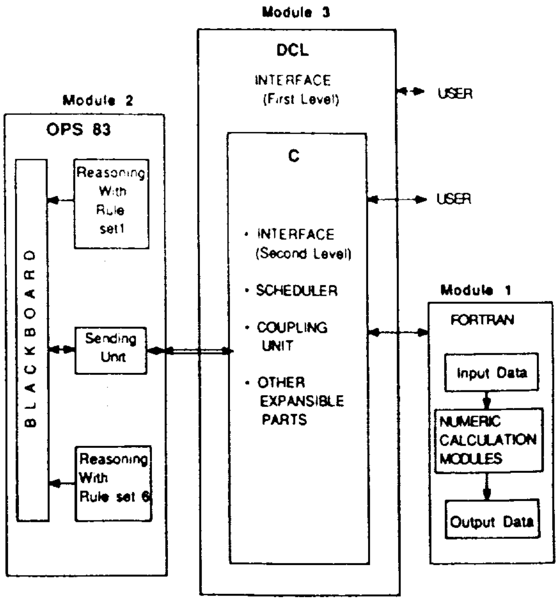


Figure 3: SYSTEM STRUCTURE OF THE EXPERT SYSTEM

TEST RESULTS

The proposed KBHO has been tested on several power systems including IEEE 14-bus, 30-bus and 118-bus test systems. The demonstration on a 14-bus system is discussed in this paper.

The objective of this study was to validate the KB support for selecting weights, misranking and masking and system partitioning while improving given performance index approach to SSA. A major loss of line for the 14-bus system causes critical contingencies leading to voltage violations. Table 6 gives the result of the ranking of contingencies in order of severity, based on the traditional performance index (PI) method (Medicherla IEEE, 1982) for voltage. The identified critical violations with and without the knowledge base (KB) are shown in Table 7. The use of KBHO leads to proper selection of appropriate weights including the identification of contingencies causing masking and misranking, thus reducing the PI lists.

Flow violations are evaluated using the PI by (Mikolinnas et al, IEEE 1981). The identified critical violations with and without KB are also shown in Table 8. The scheme is able to correctly identify appropriate weight for a given PI and at the same time reduce the effects of masking and misranking due to a given violation.

Classical Approach Using The Voltage Performance Index		Expert Aided Assisted Contingency Ranking Using The Voltage Performance Index			
Rank	PI Without KB	Masking/ Misranking Effects	Weight Selection Effects	System Partitioning	Performance Index Using KB
12*	0.5201	x	x	12*	0 1045
3*	0.0965	x	x	15*	0 0875
15*	0.0907	x	x	3	0.0700
2*	0 0373		x	2	0 0147
7	0.0350		x	7	0 0145
4	0.0158		x	4	0 0060
16	0 0146		x	14	0 0045
10	0.0114		x	10	0.0038
20	0.0075	X	x	8	0.0029
17	0.0072	X	x	5	0 0015
1	0.0047	X	x	1	0 0012
5	0.0047		x	9	0.0010
6	0.0035				
8	0.0026				
13	0.0025				
9	0.0023				
14	0.0022				
11	0.0020				
19	0.0008				
18	0.0007				

* Denotes critical contingencies

TABLE 7: IMPROVED VOLTAGE BASED PERFORMANCE INDEX (14-BUS SYSTEM)

Classical Approach Using Power Flow Performance Index		Expert System Assisted Contingency Ranking Power Flow Performance Index			
Rank	PI Without KB	Masking/ Misranking Effects	Weight Selection Effects	Rank	PI With KB
2*	4.8200	x	x	1*	4.5800
1*	4.3700	x	x	2*	3.2300
14	1.4900		x	14	2.800
6	0.1010	X	x	8	0.6070
15	0.0960	X	x	4	0.3980
9	0.0960	X	x	3	0.2800
16	0.0084	X	x	6	0.1690
17	0.0070	X	x	15	0.1650
20	0 0041	X	x	9	0.1230
13	0 0027	X	x	20	0.0535
18	0 0018	X	x	18	0 0535
19	0.0004	X	x	17	0.0164
12	-0.0292	X	x	16	0.0142
11	-0.0610	X	x	19	0.0048
8	-0.0690	X	x	13	0.0028
4	-0.1710	X	x	12	0.0231
3	-0.2500		x	11	0 0515
5	-0.4400		x	5	-0 2040
7	-0.5188		x	7	-0.3600
10	-0.9500		x	10	-0.5700

TABLE 8: IMPROVED POWER FLOW-BASED PERFORMANCE INDEX (14-BUS SYSTEM)

To test the KBHO scheme as a corrective measure for removing violations in a planning process, selection of objective functions and constraints are performed via the KB. During the effect of contingency #1 power flow violation is corrected with operating voltage limits by using the cost objective function. Other violations are similarly corrected with new operating states control and applicable objective functions identified. These results are shown in Table 9.

The KB optimization scheme is capable of selecting appropriate constraints and objectives. The scheme also guarantees convergence and improves currently used contingency screening schemes.

Contingency	Operating Range	Post Contingency Values	Corrected values	Selected Object. Function	Remarks
# 1 (Power Flow Violation)	0.000 to 0.990	1.217	0.954	Cost	Cost Objective Function was Selected to Correct Power Flow Violation
# 2 (Power Flow Violation)	0.000 to 1.080	1.225	0.982	Cost	Power Flow Constraints were Satisfied on All Circuits
# 3 (Voltage Violation)	0.925 to 1.075	0.8975	0.984	voltage	The Voltage Objective Function was Selected to Correct the Violation at bus # 10

TABLE 9: CORRECTION OF SELECTED VIOLATIONS USING CONTINGENCY-CONSTRAINED OPTIMAL POWER FLOW (14-BUS SYSTEM)

CONCLUSION

Implementation of optimization techniques in system planning involves complex interaction between human planner, numerical programs and objective system status. Appropriate selection of options relating to algorithms, objective functions, conditional constraints, parameters, etc., plays an important role in reaching system wide optimization. This paper suggests a knowledge-based approach to improving optimization techniques in system planning through a knowledge-based implementation of the options. The concept of knowledge-based hierarchical optimization in a planning process is presented to described the need for applying knowledge-based methodology to optimization consideration of system planning.

A knowledge-based implementation of the optimal power flow-based static security assessment of power systems, as an example of system planning, is presented to improve the traditional implementation of static security assessment. An expert system designed to the knowledge-based scheme of static security assessment has been described. Test results verified the feasibility of the scheme including the expert system used for implementing the scheme.

ACKNOWLEDGEMENTS

This work was supported by the United States Department of Energy, the National Science Foundation under grant ECS 8657559PYI Award and City of Los Angeles Department of Water and Power. The authors wish to thank the staff of the Energy Systems Network Laboratory, especially Mr. Anthony Akande for his assistance in the preparation of the manuscript.

REFERENCES

Alvery IKBS Special Interest Group Report, "IKBS Planning Workshop No. 5," *IEE, P. O. Box 26, Hitchin, Hertfordshire SG61SA*, UK, March 1986.

J. P. Sangiovanni and H. G. Romans, (Sept. 1987). "Expert Systems in Industry: A Survey," *Chemical Engineering Process*, V. 83, pp. 52-59.

R. K. Miller, and T. C. Walker, (1988). "Artificial Intelligence Applications in Research and Development," *SEAI Technical Publications.*

Z. Z. Zhang, G. S. Hope and O. P. Malik, (Nov. 1989). "Expert Systems in Electric Power Systems-And Bibliographical Survey,"*IEEE Trans. on Power Systems*, Vol. 4, No. 4. pp. 1355-1362.

Ph. L. Toint, (1988). "A View of Non-linear Optimization in a Large Number of Variables," *Simulation and Optimization of Large Systems, edited by A. J. Osiabacz, Clarendon Press*, pp. 83-116.

O. Alsac, Brian Scott, (May-June 1974). "Optimal Load Flow with Steady State-Security," *IEEE Trans. Power Apparatus and Systems*, Vol. PAS-93, pp. 745-751.

J. A. Momoh, (Nov. 14-17). "A Generalized Quadratic-Based Model for Optimal Power Flow," *Proceedings of the Conference on Systems, Man and Cybernetics*, in Cambridge-Massachusetts, 1989, pp. 261-267.

Thomas J. Verney,(April 27-29, 1988). "Results of EPRI Project RP-1712: Static security Analysis an Utility Demonstration," *Proceedings of workshop on Power systems security Assessment*, Ames, IOWA, pp. 135-154.

G. C. Ejebe, B. F. Wollenberg, (January/February1979). "Automatic Contingency Selection,"*IEEE Trans on PAS*, Vol. PAS-98, No. 1, pp. 97-109.

T. K. P. Medicherla and S. C. Rastogi, (Sept. 1982). "A Voltage Criteria-Based Contingency Selection Technique," *IEEE Trans. on PAS*, Vol. 101, No. 9.

T. A. Mikolinnas, B. F. Wollenberg, (Feb. 1981) "An Advanced Contingency Selection Algorithm," *IEEE Trans. on PAS*, Vol. 100, No. 2, pp. 608-617.

N90-22304

MANAGING TEMPORAL RELATIONS

Daniel L. Britt
Amy L. Geoffroy
John R. Gohring
Martin Marietta Information Systems Group
P.O. Box 1260
Denver, CO 80201-1260

ABSTRACT

In this paper we will describe various temporal constraints on the execution of activities, and discuss their representation in the scheduling system MAESTRO* . Initial examples will be presented using a sample activity to be described. We will then expand upon those examples to include a second activity, and explore the types of temporal constraints that can obtain between two activities. Soft constraints, or preferences, in activity placement will be discussed. Multiple performances of activities will be considered, with respect to both hard and soft constraints. The primary methods used in MAESTRO to handle temporal constraints will be described as will certain aspects of contingency handling with respect to temporal constraints. We will conclude with a discussion of the overall approach, with indications of future directions for this research.

INTRODUCTION

In order to describe temporal constraint handling in the scheduling system MAESTRO, it will be helpful to first discuss in general what MAESTRO schedules, what a schedule is and how it's built.

The basic schedulable entity MAESTRO deals with is called an ACTIVITY. An activity is a set of actions which, when successfully completed, accomplishes some desired goal. We call these actions SUBTASKS, and specify that an activity is an ordered sequence of non-overlapping subtasks. For example, suppose we wish to perform a spectral analysis of a portion of the upper atmosphere using a satellite-born instrument. We could call this activity ATMOS. The sequence of subtasks which make up ATMOS are listed in table 1. We must power up the instrument, perform a self-test

* MAESTRO is a proprietary product of Martin Marietta Corporation.

on its electronics, calibrate it using a known light source, repoint it, collect the data we're interested in, and then put the instrument back in stand-by mode.

Table 1. Activity ATMOS.

Subtask	Name
1	Power Up
2	Self Test
3	Calibrate
4	Repoint
5	Collect Data
6	Power Down

In addition to descriptions of activities such as the one above, we need profiles of available resources used by the activities as well as profiles of ambient environmental conditions in order to schedule these activities. These profiles describe the state of each resource or condition as a function of time. Figure 1 shows an electrical power availability profile.

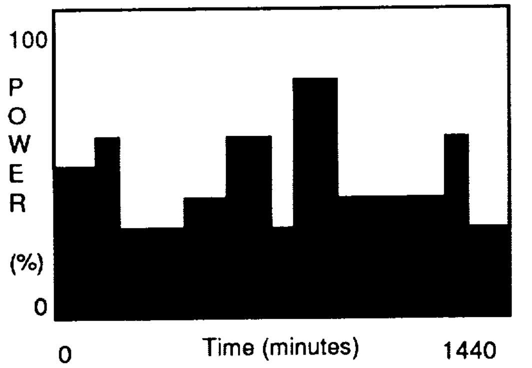


Figure 1. Electrical power availability as a function of time.

Given a set of activity descriptions and resource and conditions profiles, MAESTRO schedules by repeatedly executing what we call a select-place-update cycle. An activity is selected from among all activities requested to be scheduled. It is then placed on the schedule such that it can be executed as placed. Finally, its proposed use of satellite resources is noted on the appropriate resource availability profiles, and a calculation is performed to determine, within these new profiles, where on the schedule each remaining activity requested can be placed. MAESTRO will continue to execute these three steps, select, place and update, until there are no more activities requested to be scheduled which can be placed.

CONSTRAINTS ON SUBTASK EXECUTION WITHIN AN ACTIVITY

We will now begin to explore the various types of constraints which dictate where on the schedule our sample activity can be placed. Each of the subtasks making up the activity has certain resource and conditions requirements which must be met for the entire duration of the subtask in order for that subtask to execute. For example, the data collection subtask requires that the instrument be available, with enough electrical power to operate it in the proper mode, and that the instrument be pointed at the right area of the atmosphere, while not being pointed too near the sun. The instrument must also be in the right temperature range, and the platform must not be subject to too much vibration. Table 2 outlines these requirements for this subtask.

Table 2. Resources and conditions needed by ATMOS subtask 5.

Resource/Condition	Amount/Value
Instrument available	Yes
Power	400 watts
Target	Earth Limb
Sun Exclusion Angle	32 degrees
Temperature Min & Max	10 - 36 deg. C
Max Vibration	650 micro-g

Each of the resources and conditions listed above has an associated availability profile maintained by the scheduler. These can be used to generate lists of time windows during which a subtask can be running with respect to each requirement. The intersection of these windows determines when all resource and conditions requirements for the subtask are simultaneously met (see figure 2). In the MAESTRO system this is known as opportunity calculation. Since any number of lists of windows can be intersected, no limit is placed on the number of resources and conditions considered. Further, these lists of windows can come from any source, so a scientist can specify all those time windows during which he wants each subtask to be running. The user can restrict the performance of the whole activity to certain time windows as well.

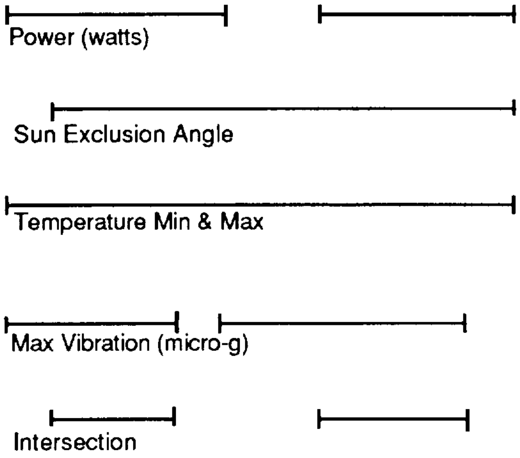


Figure 2. Time windows wherein a subtask can be "on" with respect each of four requirements, and their intersection.

The above calculation results in a clear picture of when each subtask in the activity can be running in isolation, but doesn't go far enough. Typically there are strict requirements on when each subtask can start and end relative to the placement of the others. The calibration subtask in the ATMOS activity, for example, must begin no later than 5 minutes after the self test subtask, which itself must immediately follow the power up subtask. In addition, each subtask has minimum and maximum durations specified by the scientist. The constraints listed above which determine the structure of the activity are constraints not on when each subtask can be running, but rather on when each can start and end. The two constraints used within an activity are the PRECEDES constraint and the FOLLOWS constraint. The end of the calibration subtask must *follow* its start by between 4 and 6, and must *precede* the start of the repointing subtask by at least 0 and no more than 10, for example. The complete list of these constraints is shown in Table 3.

Table 3. Subtask durations and delays for activity ATMOS.

Subtask	Duration		Delay	
	min	max	min	max
1	3	3	-	-
2	1	1	0	0
3	4	6	0	5
4	1	10	0	10
5	18	36	0	0
6	3	3	0	0

The precedes and follows constraints need not be applied only to adjacent subtask start and end points, but can constrain any two. Thus we can specify that the data collection subtask follow the calibration subtask by between X and Y. We can also limit the duration of the whole activity by placing precedes and follows constraints between the start of the first subtask and the end of the last.

CONSTRAINTS BETWEEN ACTIVITIES

Thus far we have considered only the constraints on a single activity, those which dictate its placement relative to resources, conditions and time windows, or its internal structure. Suppose, however, that in order to understand the data coming back from ATMOS in our example, the scientist needs to also get data from another instrument, in this case a solar spectrometer. This data must be taken at the same time that the ATMOS data is taken in order to correlate the results. We can describe the second activity, called SOLAR, much the way we did the first (see Table 4), but we also must specify the timing constraints between the two. We specify that the data collection subtask in the ATMOS activity must *start* and *end* at the same time as the data collection subtask in the SOLAR activity. These two new constraints, STARTS and ENDS, are similar to the

PRECEDES and FOLLOWS constraints referred to previously, and like those can have variable offsets. For example, we could specify that subtask 5 of ATMOS start between 2 and 5 minutes after the start of subtask 4 of SOLAR. Figure 3 shows the relationship between ATMOS and SOLAR.

Table 4. Activity Solar.

Subtask	Name
1	Power Up
2	Self Test
3	Repoint
4	Collect Data
5	Power Down

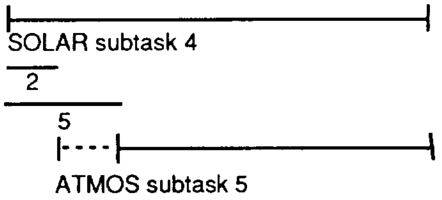


Figure 3. Temporal relationship between ATMOS subtask 5 and SOLAR subtask 4.

So far we have identified four temporal constraints - PRECEDES, FOLLOWS, STARTS, and ENDS. These dictate a relationship between a constrained entity and a constraining entity. The fifth subtask of ATMOS was the constrained entity in the example above. Specifically, the start of that subtask was constrained by the start of subtask 4 of SOLAR. All STARTS constraints will be of this nature. Similarly, all PRECEDES constraints will dictate a relationship between the end of the constrained entity and the start of the constraining entity. Table 5 shows the complete list of relationships specifiable with these four constraints.

Table 5. Relationships between constrained and constraining entities.

Constraint	Boundary of Constrained Entity	Boundary of Constraining Entity
PRECEDES	end	start
FOLLOWS	start	end
STARTS	start	start
ENDS	end	end

A fifth constraint type is necessary to fully specify possible relationships between constraining and constrained entities, the CONFLICTS constraint. Suppose we wish to specify that the calibration subtask of ATMOS will be disrupted if we try to perform the repointing subtask of SOLAR at the same time. We can indirectly represent this by causing SOLAR's subtask 3 to produce a condition, say vibration, which subtask 3 of ATMOS cannot tolerate, tracking vibration along with temperature and other conditions. More straightforwardly, we could specify that subtask 3 of ATMOS *conflicts* with subtask 3 of SOLAR. Like the others, this constraint can include variable offsets, allowing the constraining entity to block the constrained entity outside its duration.

It should be noted that the five constraint types listed above, with variable offsets, can singly or in combination express all of the relationships specified by Allen in his work on temporal constraint representation [Allen 1981]. We use a uniform interpretation of the meaning of positive and negative offsets on the constraints in MAESTRO such that a single algorithm can correctly propagate all of these constraints. An algorithm used for constraint propagation in scene understanding developed by Waltz [Winston 1984] has been modified for use in the scheduler. Given lists of time windows wherein each subtask can be running, it finds all and only those places on the schedule where each can start and end.

Notice that while the placement of ATMOS depends upon that of SOLAR, the reverse is not true. SOLAR can happen without regard to where or whether ATMOS is placed in order to achieve its own objectives. This is called a ONE-WAY or unidirectional constraint. If we wish to only perform SOLAR when it can support ATMOS, we can specify that it be a TWO-WAY or bidirectional constraint. A two-way constraint specifies a temporal relationship between two activities, and requires that neither can be scheduled without the other. Two-way constraints are more difficult to deal with than one-way constraints in MAESTRO. The select-place-update cycle described previously is designed to place a single activity, given complete knowledge (through opportunity calculation and temporal constraint propagation) of all possible placement options for that activity with respect to the current partial schedule. The existence of a two-way constraint precludes knowing all possible placements, since for each activity the position on the schedule of its constraining entity is not fixed. In MAESTRO we deal with this by placing more than a single activity on the schedule on each scheduling cycle. We calculate opportunity individually for each activity in a set of mutually related activities, then allow the constraint propagation algorithm to run on all activities in that set simultaneously. We call the set of mutually related activities a related set.

If the activities in the related set are independent of one another except for the temporal constraints putting them in the same set, the constraint propagation algorithm again finds all and only those places on the schedule where each subtask can start or end. However, if the activities share resource use or produce conditions which affect one another, this knowledge cannot be obtained with this algorithm. This allows the possibility that the placement of a related set will fail, necessitating backtracking. We have ruled out many placement possibilities that won't work and so can try various choices within those placements we think might work, and can apply various heuristics which are aimed at making sure each placement is significantly different from the last, but trial and error is involved if the activities share resources or have a producer-consumer relationship.

The use of related sets has been expanded in MAESTRO to include more than dealing with two-way temporal constraints. In our example involving ATMOS and SOLAR, we may have only a one-way constraint (ATMOS constrained by SOLAR), but may consider it much more important to schedule ATMOS than to schedule SOLAR. In this case we would like the two to be considered as both being important, and further would like the scheduler to only place SOLAR where it can support ATMOS. The related sets facility allows us to do this.

To this point we have considered only those situations wherein the placement of a subtask in one activity dictates where on the schedule a subtask in another activity can be placed. It is often desirable to specify an absolute time which constrains the start or end of a subtask in an activity. We also may wish to relate subtask placement to that of some event which will happen at various times but is not under control of the scheduler. Both these situations are handled in MAESTRO the same way we deal with one-way constraints between activities not in the same related set. Thus while constrained entities are always subtasks, constraining entities can be subtasks, events or absolute times. A constraint on an activity is represented as a constraint on the first or last subtask of that activity.

SOFT CONSTRAINTS

All of the constraints dealt with previously have been hard constraints, which must be satisfied in order for the constrained activity to execute. Another class of temporal constraints are soft constraints, or preferences. These guide the scheduler in placing activities on the schedule where it is most desirable, according to a scientist, platform manager, etc. Unlike hard constraints, however, these can be ignored if necessary to get things done. For example, it may be desirable to get data from ATMOS as near to noon, GMT, as possible, but not really necessary.

In MAESTRO, several types of soft constraints are representable. There are soft constraints which guide placement of a whole activity, called general preferences. Loading strategies are a type of general preference which guide placement of the activity with respect to the time period being scheduled. Front-loading, getting things done as early as possible, is a particularly attractive loading strategy in that activities scheduled earlier have a better chance of completing before something happens that might interfere with their completion. Various other loading preferences are supported. Other general preferences guide the structuring of the activity when there are variable durations for subtasks and/or variable delays between them. We can, for example, request maximizing durations and minimizing delays within the context of the loading strategy used for the activity being scheduled.

There are times when we wish to specify a preference which overrides these general preferences. We can, for example, ask the scheduler to place an activity where a particular subtask duration is maximized, regardless of where on the timeline that placement is found. This is called a specific preference, and is attended to in MAESTRO before any general preferences. Another type of specific preference guides the scheduler in placing activities either near to or far from other activities, events, or timepoints. Currently MAESTRO allows the specification of only one specific preference per activity, as the simultaneous satisfaction of two or more specific preferences is ambiguously defined.

Occasionally it happens that soft constraints on activity placement are at odds with the allowable placements as dictated by hard constraints, and these can interact in interesting ways. If a user requests that subtask 2 of ATMOS be placed as early as possible but the data collection subtask can only be placed late, the effect will be to stretch the activity out, maximizing delays between subtasks. In order to avoid this the scheduler under certain conditions will ignore general loading preferences and will intelligently order the application of general duration and delay preferences when one or more subtasks in the activity are highly constrained.

The approach taken in MAESTRO to scheduling typically yields "good" schedules, those which adhere to all hard constraints, pay attention to soft constraints when possible, and "get a lot done". It is sometimes desirable to ignore all preferences and just place activities randomly in an attempt to find a better schedule by generating several and choosing the best one, so MAESTRO has a random placement option. Using this option all hard constraints are still met, but a different schedule is generated each time the scheduler is run, allowing various activity placement combinations to be explored. Also, a user may wish to personally place some or all of the subtasks making up an activity, and this option is under implementation.

MULTIPLE PERFORMANCES

So far in this paper we have treated activities as if they were designed, scheduled, performed once and then forgotten. Typically, however, a user will want an activity to be performed many times. It can be the case that if an activity is not performed at least N times, it is not worth doing at all. Thus in MAESTRO a user can specify a minimum success criterion, a least number of performances acceptable to him. The scheduler uses these criteria in deciding which activity or related set to schedule next. The requirement to schedule multiple performances of activities makes scheduling more complex with respect to temporal constraints. If two temporally related activities each request several performances, and if there are variable offsets between them, it may be ambiguous which performances constrain which (see figure 4). MAESTRO maintains an interpretation of the relationships between performances such that constraints are never violated.

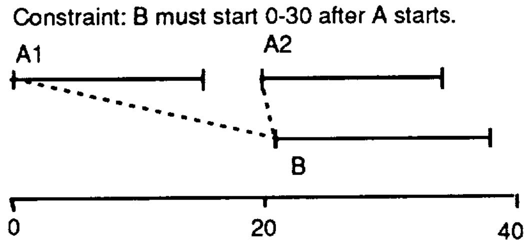


Figure 4. Multiple performances which may constrain others ambiguously.

A user may dictate that his experiment not be repeated more often than once every four hours, which introduces the idea of minimum performance separation. This is treated in MAESTRO somewhat like a one-way constraint. It is worthy of note that a negative performance separation, or overlap, is allowed by MAESTRO. A crewman performing an experiment in the lab module on Space Station Freedom may wish to begin preparation of a second sample before finishing the data analysis on the first, for example.

Typically when two activities are related by a temporal constraint it is required that one performance of the constraining activity be scheduled with one performance of the constrained activity. It may, however, be desirable to perform an activity once each third (nth) time that another is performed. This requirement is called a constraint arity. Facilities in MAESTRO for dealing with constraint arities other than one-to-one are not yet complete.

CONTINGENCY HANDLING

We have discussed a number of issues dealing with the generation of a schedule and the management of temporal relations involved. This scheduling is part of an ongoing operations environment wherein the assumptions upon which a completed schedule was based can change at any time, making the schedule invalid. It is preferable in most cases to alter the existing schedule rather than generating a whole new schedule for the time period encompassing the changes. Making changes to an existing schedule in response to changes in requirements, resource availabilities, etc., is known as contingency handling. One requirement levied on contingency handling processes is that they produce a modified schedule in which no temporal constraints are violated.

There are three aspects to contingency handling. One is simply scheduling; a late-arriving request to schedule an activity may only require that the activity be scheduled, with no other schedule changes. We have previously explored many aspects of temporal relations in scheduling. Another aspect is unscheduling, wherein a performance of an activity is removed from the schedule entirely in order to reduce resource usage, allow another activity to fit, or because a user no longer wishes to perform the activity. If an activity which constrains others is unscheduled, those others must be unscheduled as well.

The third aspect of contingency handling involves activities which have already begun to be executed but which cannot complete as scheduled. This may happen as a result of resource or conditions changes which become known only after the activity has begun, or in order to fit a high-priority activity on the schedule in response to a last-minute request. In this case it is desirable to make use of various characteristics of the activity to be interrupted and attempt to find a way to continue the activity. It may be possible to switch to usage of a resource other than that which was preempted by the contingency, leaving the activity structured the same as before. The subtask which was interrupted may be such that it can be continued after a short interruption with no ill effect, or it may be possible to begin at the start of that or an earlier subtask again after a pause, not beginning the whole activity again. Also, the rest of that subtask may not be necessary, as would be the case with a long data collection subtask during which more data was collected than required, allowing the activity to be continued by going immediately to the next subtask.

In each of these cases any temporal constraints between interrupted activities must be satisfied, possibly causing other activities to be interrupted, which may themselves allow restructuring. MAESTRO handles these situations by automatically generating activity descriptions which vary from the initial descriptions in ways allowed by the activity definition. These variant activities are called alternate models. It is assumed in MAESTRO that these alternates will satisfy the same temporal constraints as the initial model would, though in the real world that would not always be the case. Several versions of the MAESTRO scheduling system exist, and the facilities for handling these realtime schedule alterations in the ways explained above do not exist as described in all versions. For a more complete discussion of issues related to contingency handling, see Britt [1988a] and [1988b].

CONCLUSION

As is readily apparent from the preceding discussion, the handling of temporal constraints in scheduling is a formidable task. We have in this paper examined the ways in which the MAESTRO scheduling system deals with various types of constraints. These include resource and conditions constraints, windows during which subtasks can be running, constraints on the internal structure of activities, hard constraints between activities and other schedule entities, soft constraints or preferences in activity placement, and constraints between performances of the same activity. We briefly touched upon issues regarding contingency handling.

The approach taken by the designers of MAESTRO is to design solutions specifically for the problems in the domain, rather than trying to fit a predetermined solution paradigm to these problems. This results in a hybrid system making use of various methods and techniques as they are proven to work [Geoffroy 1990]. Proven techniques include object-oriented design, use of opportunity-calculation and constraint propagation algorithms to minimize backtracking (by getting optimal solutions to relevant subproblems at each step), use of user-derived heuristics such as front-loading, and a control structure that allows dealing with a related set of activities when appropriate. This approach to scheduling research is made feasible at least in part by use by the design team of a powerful and flexible software development environment supported by the Symbolics LISP Machine.

There is much yet to be done to complete the temporal constraint handling facilities in MAESTRO. Constraint arities other than one-to-one need to be dealt with more completely. The scheduler can be enhanced with the addition of smarter selection, placement and contingency heuristics. There are ways not yet implemented to deal with multiple specific preferences. User selection of the placement of individual subtasks is not complete, and the creation of alternate models of the same activity, which one version of MAESTRO performs, must be incorporated with the other capabilities previously described. This is by no means a complete list of scheduler enhancements that could be undertaken.

One effort that is anticipated to have enormous payoff, if it can be done, involves changes to the temporal constraint propagation algorithm itself. As explained above, backtracking is currently necessary in those cases where subtasks which can overlap also share use of constraining resources, as the scheduler cannot determine how those overlaps will affect resource availabilities given the variations possible in subtask placement. We hope soon to implement an algorithm similar to that which currently exists but with a major difference. The new algorithm will make use of information about possible subtask overlaps, and the increased resource use incurred, as well as the information we now use concerning when individual subtasks can be running, to find all and only those times when each of a group of possibly overlapping subtasks can start and end. The existing algorithm gives us this information for non-overlapping subtasks. Given this information about overlapping subtasks, the scheduler will be capable of scheduling sets of related activities without backtracking (trial and error). It will thereby be able to make full use of preferences in activity placement as well. Though it is not certain as yet that this calculation is possible, or computationally feasible, our experience with the current algorithm suggests that it is both. We intend that this and other new capabilities be installed in MAESTRO in the near future.

REFERENCES

Allen, J.F., (1983). "Maintaining Knowledge About Temporal Intervals." *Communications of the ACM*, 26(11), 832-843

Winston, P.H., (1984). *Artificial Intelligence*, pp 66-72. Reading, MA: Addison-Wesly

Britt, D.L., Geoffroy, A.L., & Gohring, J.R. (1988a). "Contingency Rescheduling of Spacecraft Operations." *Telematics and Informatics*, 5(3), 187-195.

Britt, D.L., Geoffroy, A.L., & Gohring, J.R. (1988b). "The Impact of the Utility Power System Concept on Spacecraft Activity Scheduling." *The Proceedings of the 23rd Intersociety Energy Conversion Engineering Conference*, v. III.

Geoffroy, A.L., Gohring, J.R. & Britt, D.L. (1990). "The Role of Artificial Intelligence in Scheduling Systems." (In preparation).

Diagnosis/Monitoring

N90-22305

NASA GROUND TERMINAL COMMUNICATION EQUIPMENT AUTOMATED FAULT ISOLATION EXPERT SYSTEMS

Y. K. Tang and C. R. Wetzel

Ford Aerospace Corporation
7375 Executive Place
Seabbrook, MD 20706

ABSTRACT

This paper describes the prototype expert systems that diagnose the Distribution and Switching System I and II (DSS1 and DSS2), Statistical Multiplexers (SM), and Multiplexer and Demultiplexer systems (MDM) at the NASA Ground Terminal (NGT). A system level fault isolation expert system monitors the activities of a selected data stream, verifies that the fault exists in the NGT and identifies the faulty equipment. Equipment level fault isolation expert systems will be invoked to isolate the fault to a Line Replaceable Unit (LRU) level.

Input and sometimes output data stream activities for the equipment are available. The system level fault isolation expert system will compare the equipment input and output status for a data stream and perform loopback tests (if necessary) to isolate the faulty equipment. The equipment level fault isolation system utilizes the process of elimination and/or the maintenance personnel's fault isolation experience stored in its knowledge base. The DSS1, DSS2 and SM fault isolation systems, using the knowledge of the current equipment configuration and the equipment circuitry, will issue a set of test connections according to the predefined rules. The faulty component or board can be identified by the expert system by analyzing the test results. The MDM fault isolation system correlates the failure symptoms with the faulty component based on maintenance personnel experience. The faulty component can be determined by knowing the failure symptoms.

The NGT fault isolation prototype is implemented in Prolog, C and VP-Expert, on an IBM AT compatible workstation. The DSS1, DSS2, SM, and MDM equipment simulators are implemented in PASCAL. The equipment simulator receives connection commands and responds with status for the expert system according to the assigned faulty component in the equipment. The DSS1 fault isolation expert system was converted to C language from VP-Expert and integrated into the NGT automation software for offline switch diagnoses.

Potentially, the NGT fault isolation algorithms can be used for the DSS1, SM, and MDM located at Goddard Space Flight Center (GSFC). The prototype could be a training tool for the NGT and NASA Communications (Nascom) Network maintenance personnel.

1.0 INTRODUCTION

This section will describe the background, problem, objective, and scope of this paper.

1.1 Background

The NGT is located with the ground terminal portion of the Tracking and Data Relay Satellite System (TDRSS) at White Sands, New Mexico. The primary role of the NGT is to serve as the interface for communication between the TDRSS and NASA facilities at Goddard Space Flight Center (GSFC) and Johnson Space Center (JSC). The primary functions of the NGT are data transport, data quality monitoring, and line outage recording and data rate buffering.

In order to meet the future (early 1990's) workload of multiple Tracking and Data Relay Satellite (TDRS) support and user support requirements, an NGT Automation (NGTA) project was completed at the end of 1989 (GSFC, STDN No. 528, 1986). The NGTA provides the capabilities to automatically configure the major NGT subsystems and to monitor their health and status using high-speed scheduling messages from the Network Control Center (NCC). The major communication equipment includes the Distribution and Switching System I and II (DSS1 and DSS2), Statistical Mulitplexers (SM) and Multiplexer and Demulitplexer systems (MDM) as depicted in Figure 1. They are controlled and monitored by the DSS1 Interface Processor (IP), SM and DSS2 Interface Processor (SMD IP), MDM Automatic Control System (MACS) and NGT Control and Status System (NCSS) as depicted in Figure 2.

The NGT communication equipment troubleshooting is done manually, although much of the equipment health status is monitored automatically. The manual troubleshooting requires a skilled technician and is time consuming. An automated fault isolation system will significantly reduce the equipment down time and the required technician skill level. The rule based expert system technique has been used to assist fault isolation tasks for various GSFC supported projects (Erikson and Hooker, 1989; Luczak, et al., 1989; Lowe, et al., 1987).

1.2 Problem

The communication equipment at NGT was built in the late 1970's or early 1980's. This equipment does not allow the internal signal status to be monitored remotely. For example, the DSS1 sends to the DSS1 IP the data and clock present status at the input and output ports and also sends the result of the comparison of the input and output port signals. Only this status information received by the IP is available for the automated fault isolation process. There is no status available between the input and output ports. The internal signal monitoring capabilities are also limited for the DSS2, MDM, and SM. With limited status information, it is not possible to directly identify a faulty LRU (usually a circuit board) within the equipment.

NASA is building the Second TDRS Ground Terminal which will replace the NGT functions during the middle 1990's. It is not cost effective to enhance the NGT communication hardware to provide more internal monitoring capabilities for the fault isolation purpose. Therefore, the proposed automated fault isolation system shall use only the existing computer hardware capability and shall not impact the performance of the other functions.

With these limitations, it is a challenge to develop a low cost automated fault isolation system for the NGT in a timely manner. The specific objectives of the NGT automated fault isolation prototype follow.

1.3 Objectives

The major objective of this study is to develop an automated fault isolation system prototype using a rule based expert system to prove the feasibility of building a low cost fault isolation system that meets all the restrictions as previously described. The secondary goal is to eventually convert the prototype to an operational system. The prototype can also be used for the following:

- to explain the fault isolation approach and methodology as a training tool
- to verify the methodology during its development
- to identify the operator interface requirements.

1.4 Scope

The fault isolation prototype was developed for the DSS1, DSS2, MDM, and SM. The interface processors for the equipment were not included.

2.0 APPROACH

The NGT automated fault isolation concept takes the top down approach. The system level fault isolation expert system will first verify that the fault indeed occurred in the NGT and will identify the faulty equipment. It will then invoke an equipment level fault isolation expert system to identify the faulty component at the LRU level. There is an equipment level fault isolation expert system for each piece of equipment.

The system level fault isolation expert system will compare the equipment input and output status for a data stream and perform loopback

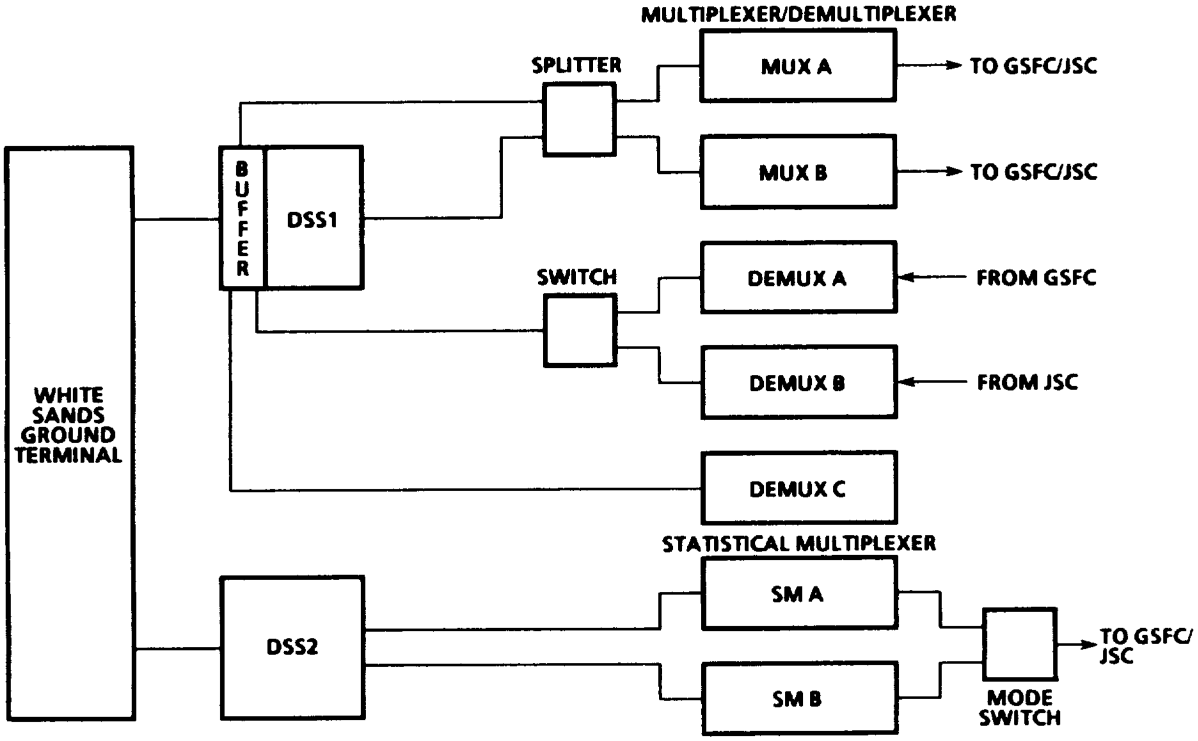


Figure 1. The NGT Communication Equipment

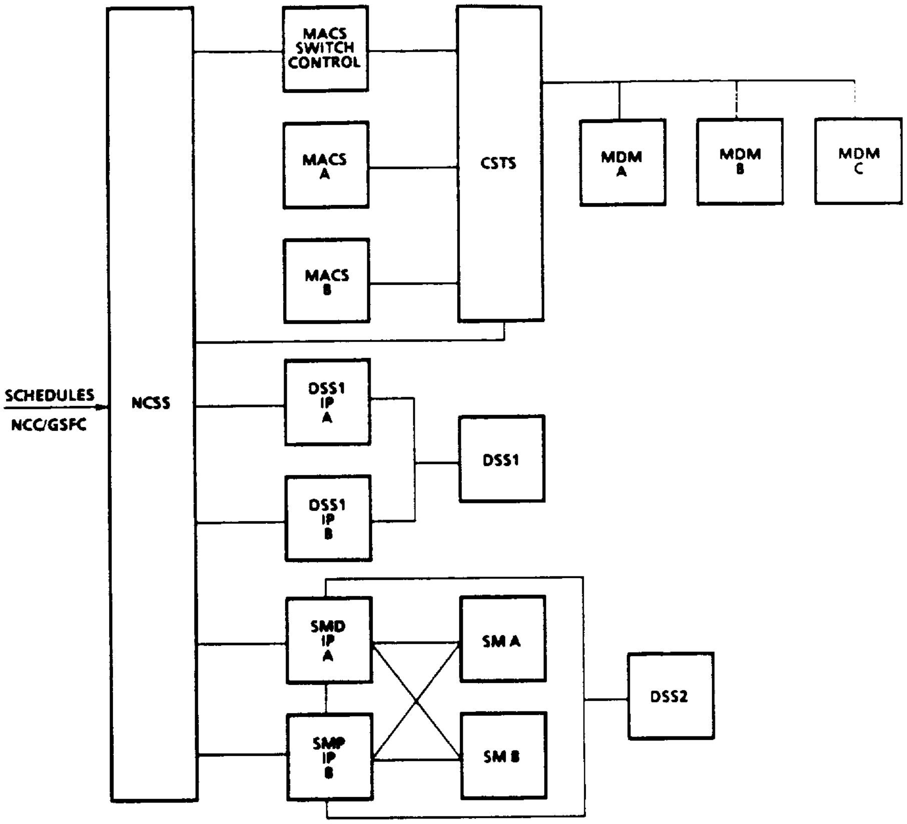


Figure 2. The Control and Monitoring of the NGT Communication Equipment

tests (if necessary) to isolate the faulty equipment. The equipment level fault isolation system utilizes the process of elimination and/or the maintenance personnel fault isolation experience stored in its knowledge base. The DSS1, DSS2 and SM fault isolation systems, using the knowledge of the current equipment configuration and the equipment circuitry, will issue a set of test connections and analyze test results according to the predefined rules.

For instance, a test data path can be chosen so that only one component on the faulty data stream is not shared by both paths (it is substituted by another component which is known to be good). If the data path is good, it can be concluded that the component not used for the test is bad. Otherwise, another component in the faulty data stream is bad and more tests are required. Using this substitution and elimination method for all components in the faulty data stream, the faulty component or board can be identified. The MDM fault isolation system correlates the failure symptoms with the faulty component based on the maintenance personnel experience. The faulty component can be determined by knowing the failure symptoms.

The system level fault isolation software can reside in the NCSS computer since it has the activity status of all equipment in the data stream. The equipment level fault isolation system can be distributed to the corresponding IP which has the capability to monitor the data stream activity status (i.e. clock or data present), and to issue test commands for additional information. The rules to compare the equipment status and to determine test cases are simple. The additional code allocated to each IP will not impact the NGTA performance. This approach provides an efficient method to achieve the NGT automated fault isolation goals under the restrictions previously described. The detailed algorithms to identify the faulty equipment and component have been prototyped and the results are presented in the following paragraphs.

3.0 FAULT ISOLATION ALGORITHMS

Both the system level and equipment level fault isolation system prototypes were developed. For each equipment level fault isolation system, an equipment simulator was built to receive test commands and to respond with status messages according to the assigned failure.

3.1 System Level Fault Isolation

The NGTA data base in the NCSS stores the data stream service configuration and scheduling information which is available to the expert system. From this, the expert system can find all the equipment used to support the data stream. The expert system will use the information for fault isolation from any equipment in the data stream that may cause an alarm or complaint. This system level NGT fault isolation can be initiated by the operator.

The system level fault isolation expert system will isolate the faulty equipment in the NGT if the data stream status is good at the equipment input port and is bad at the output port. The system will conclude that there is no fault in the NGT if all the status along a data stream in the NGT are bad and if the service is during the spacecraft acquisition, reacquisition, ground equipment reconfiguration, service-to-service handovers or first several minutes of service. During these periods, the data stream is not stable (Miksell, et al., 1987).

The system will configure the NGT ground equipment to perform loopback tests if the input data stream status is good and the output status is not available. Data stream signals at the MDM Output Controller (OC) output port, Nascom's Domestic Satellite (DOMSAT) downlink, or SM transmitter output are available for loopback tests through switching. If the loopback test shows the data stream is good, the fault is not in the NGT.

The system level fault isolation system will invoke the equipment level fault isolation system to identify the fault to the LRU level. The following paragraphs describe the equipment level fault isolation principles.

3.2 Distribution and Switching System I

The DSS1 provides buffering and switching for 192 digital low-data rate signals that pass through the NGT. The design of the DSS1 is based on a Clos-type nonblocking switch array that consists of stage A, B, and C cards. One test card and 8 input cards are also used to monitor switch status and to generate test data.

The Input, A, B, and C cards consist of decoders and selectors (4 to 1 selection). The health of a card is determined by testing the health of the decoder and selector used for a connection. The status of a selector can be determined by making a new connection that uses a different selector but otherwise the same components in the A, B, and C cards as the faulty connection. This can be done since a decoder controls a group of selectors. If the new connection is good, the selector on the faulty connection path is bad. Otherwise, another component on the path is bad and the decoder needs to be tested next. The new connection shall consist of all new components except the decoder. If the new connection is bad, the decoder is bad. Otherwise, the decoder and selector of another card on the connection path shall be tested. Through the process of substitution and elimination the faulty component and the faulty card can be determined.

The expert system, with the knowledge of switch circuitry and the current connections, will find the proper free ports to support tests, issue commands to conduct the tests, and interpret the test results.

3.3 Distribution and Switching System II

The DSS2 consists of 48 data/clock pair inputs and 40 data/clock outputs. The major line replaceable modules of the DSS2 are as follows: input module, switch module, multiplexer module, output module, peripheral electronics assembly, and computer assembly.

The clock and data status are monitored at the input module and the output module. An example of an algorithm to isolate the faulty module if the data status is good at the input but bad at the output follows.

A connection test is made by connecting the same input to a proper output such that the faulty connection and test connection share the same switch module and the same switch Large Scale Integration (LSI) circuitry in the switch module. If the new connection is good, the faulty module is either the multiplexer module or the output module. Otherwise, the faulty module is either the input module or the switch module.

To distinguish whether the fault is the input module or the switch module, another test is required. The new test will use the same input module and switch module as the faulty connection, but will select an output that uses a different switch LSI in the same switch module. If the new connection is good, the switch LSI is faulty, otherwise the fault is either the input module or the fan-out board in the switch module. More tests are needed to complete the fault isolation process.

The DSS2 fault isolation expert system has the knowledge base and rules to perform the process of elimination as previously described and will isolate the faulty module or modules.

3.4 Multiplexer and Demultiplexer

This section describes the fault isolation algorithms for the multiplexer and demultiplexer portions of the MDM separately.

3.4.1 Multiplexer

The Multiplexer systems at the NGT consist of 100 Input Terminal Units (ITU) and Triple redundant Output Controllers (OC). The Multiplexer system is fully redundant. Both prime and alternative systems process and transmit the composit data streams simultaneously. The fault isolation processes includes loopback tests and direct interpretation of the failure symptoms.

Only the input clock status at the ITU for a data stream in the Multiplexer system is monitored. Loopback tests are required to determine that the fault is indeed in the Multiplexer as descried 3.1. If the fault is determined to be in the Multiplexer, and GSFC or JSC experience problems on all the data streams from the NGT, the fault is in the OC. If only one data stream has problems, the fault is in the corresponding ITU. There are three boards in an ITU and 7 boards in an OC. Further fault isolation to the board level requires the knowledge that associates the failure symptoms and/or alarm messages to the most likely faulty board.

The expert system, with the knowledge of the multiplexer configurations from the service schedules and the failure symptom correlations from maintenance personnel experiences, will configure the loopback tests and interact with the operator to isolate the faulty board.

3.4.2 Demultiplexer

A demultiplexer consists of one Input Controller (IC) and 30 Output Terminal Units (OTU). There are two Demultiplexer systems at the NGT to process the signals from GSFC and JSC separately. The third one is used as a spare and/or to support a recorder playback function.

The Demultiplexer fault isolation is based on the error messages the demultiplexer generates and the failure symptoms the operator observes. Instead of loopback tests, the spare demultiplexer is used to support fault isolation. The composite data stream from GSFC or JSC will be routed to the spare unit and demultiplexed, and the output data stream status will be monitored. If the data stream is good then the fault is in the NGT. There are three logic cards in an OTU and four logic cards in an IC. If only one data stream has problems, the fault is in the associated OTU. If all the data streams have problems, the fault is in the IC. Based on the failure symptoms the most likely faulty logic card can be isolated.

3.5 Statistical Multiplexer

The SM at the NGT consists of a transmit section and a receive section. Four input ports are available for the transmit section and four output ports are available for the receive section. A spare SM at the NGT is available for backup and fault isolation support.

There is a receive module and a transmit multiplex module within the transmit section. There are demultiplex modules, a patten detector module, frequency synthesizers, and error drivers within the receive section. The transmit/receive module and high speed data driver boards are shared by both sections. During normal operations, the composite data stream output from the transmit section is looped back to the receive section of the same unit, and the composite data stream from the SAT is looped back to the receive section of the spare unit for data status monitoring.

When the receive sections of both units indicate data or clock loss, the fault is in the transmit section. The service will be restored by switching the input data streams to the spare unit. The faulty module will be determined after the service is over and during equipment free time.

For instance, a transmit section fault can be determined by feeding the test data to all four input ports and looping the composite data to the receive section of the demutiplexer. If all ports at the receive section lose clock and data, the fault is in the transmit/receive module. If only one port at the receive section loses clock and data, the fault is in the corresponding transmit multiplex module. By examining the failure impacts the faulty module can be identified.

The expert system will interact with the maintenance personnel to setup the test data generator and to configure the loop back tests. The expert system will monitor the data stream status and determine the faulty module.

4.0 PROGRESS AND PROTOTYPE USAGE

The system level fault isolation expert system was implemented in dBASE III to take advantage of a relational file structure for storing the service schedules and configuration information, and for the ease in data input. The DSS1 fault isolation expert system was implemented in VP-Expert, a rule-based expert system development tool (Sawyer, et al., 1987). The DSS2, MDM, and SM fault isolation expert systems were written in Turbo Prolog. The simulators for the equipment were written in Turbo Pascal. All software was implemented on an IBM AT compatible workstation. The system level and the DSS1 equipment level fault isolation programs were converted to C language for possible integration with the operational NGTA software.

The user can test the prototype by assigning one communication equipment fault at the board or module level, by configuring a service identifying the equipment involved and ports used, and by starting data transmission. The prototype will issue the appropriate error messages to indicate a system failure, then the user can initiate the fault isolation procedure. The system level fault isolation system will display the service configuration graphically along with all the available color coded clock and data status. The faulty equipment will be highlighted in red.

The equipment level fault isolation system will be invoked by the system level fault isolation system. The system will display a flow diagram to show all the modules in the equipment needed to support the faulty service along with the available status. The prototype will also show the test data path and status on the same diagram but in a different color. The components shared by both paths can be seen clearly. The component determined to be healthy after a test will turn green. The user is able to observe the process of elimination step by step until the faulty board or module is isolated. The test connections and results are also explained in the message window. The fault isolation result can be compared to the fault initially assigned to verify the success of the fault isolation system.

A series of tests were performed to debug the software and to verify the fault isolation algorithms. The fault isolation prototype successfully demonstrates the automated fault isolation capabilities. The DSS1 fault isolation algorithm was actually tested at the NGT. Chips in the A card, Input card and Test card were purposely damaged to create faults. A set of test connections were issued manually according to the fault isolation algorithm. By analyzing the connection test results, the faulty boards were correctly identified.

5.0 CONCLUSIONS AND RECOMMENDATIONS

The NGT fault isolation prototype does demonstrate the feasibility to apply expert system techniques to perform the automated fault isolation tasks with minimum operator intervention. The conclusions of the prototype implementation are as follows:

- The prototype proves that it is feasible to develop automated fault isolation algorithms for the NGT communication equipment.
- The fault isolation algorithms are simple and effective. There is no hardware enhancement required to implement the algorithms.
- The fault isolation system software can be distributed and integrated into the NGTA subsystems to perform automatic test configuration, status monitoring and interpretation, and fault isolation with minimal impacts to the system response time.
- The prototype can be used as a training tool to explain the fault isolation algorithms.

The prototype demonstrates the feasibility and cost-effectiveness to add the automated fault isolation capabilities to the NGT. It is recommended that the fault isolation capabilities be implemented at the NGT and other NASA facilities with similar equipment such as Nascom and Second TDRS Ground Terminal.

REFERENCES

Erikson, C., and Hooker, P., "Tracking & Data Relay Satellite Fault Isolation & Correction Using PACES: Power & Attitude Control Expert System," 1989 Goddard Conference on Space applications of Artificial Intelligence, May 1989.

Lowe, D., et. al., "FIESTA: An Operational Decision Aid for Space Network Fault Isolation," 1987 Goddard Conference on Space applications of Artificial Intelligence, May 1987.

Luczak, E., et. al., "REDEX: The Ranging Equipment Diagnostic Expert System", 1989 Goddard Conference on Space application of Artificial Intelligence, May 1989.

Miksell, S., et. al., "Network Fault Diagnosis: Knowledge Representation Using Parallel Rea-soning", 1987 Third Annual Expert Systems in Government Conference Proceedings, October 1987.

"NASA Ground Terminal (NGT) Automation Project Plan," STDN No. 528, September, 1986, Goddard Space Flight Center.

Sawyer, B., et. al., "VP EXPERT", PAPERBACK SOFTWARE INTERNATIONAL, 1987.

AUTONOMOUS POWER EXPERT SYSTEM

Jerry L. Walters, Edward J. Petrik,
Mary Ellen Roth, and Long Van Truong
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135

and

Todd Quinn and Walter M. Krawczonek
Sverdrup Technology, Inc.
NASA Lewis Research Center Group
Cleveland, Ohio 44135

ABSTRACT

The Autonomous Power Expert (APEX) system has been designed to monitor and diagnose fault conditions that occur within the Space Station Freedom Electrical Power System (SSF/EPS) Testbed. The APEX system is being developed at the NASA Lewis Research Center by the Space Electronics Division (SED) in conjunction with the Space Station Directorate and Power Technology Division (PTD). APEX is designed to interface with SSF/EPS testbed power management controllers to provide enhanced autonomous operation and control capability.

The APEX architecture consists of three components: (1) a rule-based expert system, (2) a testbed data acquisition interface, and (3) a power scheduler interface. Fault detection, fault isolation, justification of probable causes, recommended actions, and incipient fault analysis are the main functions of the expert system component. The data acquisition component requests and receives pertinent parametric values from the EPS testbed and asserts the values into a knowledge base. Power load profile information is obtained from a remote scheduler through the power scheduler interface component.

This paper will discuss the current APEX design and development work. Operation and use of APEX by way of the user interface screens will also be covered.

INTRODUCTION

The APEX prototype system was designed as a high-level advisor for diagnosing faults in subsystems of the SSF/EPS testbed. A hierarchy of conventionally programmed controller computers reside between the symbolically programmed APEX system and the testbed subsystems (Wright, et al. (1989)). Prototype development work, for determining the design and requirements for APEX, was based on the Power Distribution Control Unit (PDCU) subsystem (Truong, et al. (1989)). APEX is currently interfaced to the PDCU subsystem controller and data communications has been established over a serial link. Ethernet communications is also available and future plans include obtaining parametric data values over the Ethernet remotely to development workstations and locally to a delivery workstation.

The APEX system consists of a rule based expert system and two interfaces that acquire data from the testbed and a remote power scheduler program. Domain knowledge from human experts has been acquired and coded into rules and stored in a knowledge base along with a model of the domain. Diagnostic rules for other SSF/EPS subsystems are to be added as Ethernet communication is established with other subsystem controllers.

Information required to diagnose faults is obtained through APEX testbed and scheduler interfaces. Parametric data values are requested from the PDCU subsystem controller by the testbed interface. Only pertinent parametric data values as determined by knowledge about expert troubleshooting techniques are requested from the PDCU subsystem controller.

Load profile information is read by the scheduler interface. Heuristics are applied to the load profile information to determine recommended actions when faults occur. Recommended actions are based on load profile information such as priorities of the loads, duration of each load, how much power the loads require for their durations, and the amount of available power from the sources subsystem.

Rule Based Expert System

Design of the expert system is based on a model which consists of objects organized into frames. This combination of objects and frames represents an integration of object oriented programming and frame based knowledge representation. The frames form a network which correspond to the PDCU subsystem. The frame representation of the subsystem contains connectivity information about the devices in the subsystem and how objects relate and have inheritance to other objects.

Fault identification is done in two phases: (1) fault detection and (2) fault isolation. Forward and backward chaining rules emulate expert reasoning necessary to detect and isolate faults. Fault detection monitors parametric values of the electrical power system to determine if the system is operating correctly. The parametric values are power, voltage, current, and status. The load profile from the remote power scheduler program contains information about expected operating conditions for each load. The APEX system determines expected analog values, for each PDCU analog test point, based on each loads scheduled operating condition. The expected test point values are compared to measured parametric values from the testbed to detect faults.

Three different types of faults that can be detected: (1) inconsistent, (2) active, and (3) incipient. Inconsistency faults occur when two or more data values give conflicting information. Active faults are detected when measured values are higher or lower than the expected values within a defined tolerance. Single and multiple active faults can be detected. Incipient faults are detected by monitoring a history of data values that identify trends toward tolerance limits. Trends are detected by statistical inference based on correlation and regression analysis of historical data. All faults are detected by forward chaining inference. Once a fault is detected, domain specific troubleshooting knowledge is referred to and backward chaining is initiated to isolate faults.

Probable causes are identified by the fault isolation phase. Rules based on knowledge acquired from domain experts are categorized by frames and associated with classes of faults. Backward chaining is initiated on the appropriate frame(s) of rules to identify probable causes. Organizing the rules with frames prevents unnecessary chaining on inappropriate rules. In some cases, more than one probable cause is displayed. When more than one probable cause is displayed the causes are shown to the operator in the order of highest to lowest probability. Justification is available to the operator to explain the reasoning process for each probable cause. Justification is obtained from the expert system from a trace back of the backward chaining rule firing. The trace back retrieves the premises of each rule that fired during backward chaining. Functions written in Lisp, translate the rule premises written in an expert system shell language, into English. The English is then displayed as a natural language explanation of the reasoning process leading to probable cause conclusions.

A recommended action feature suggests what should be done to correct the fault. The APEX system considers information such as the severity of the fault and priority of the loads in recommending the action that should be taken to correct, bypass or temporarily tolerate the fault.

Hardware and software being used for the development of APEX are Texas Instruments Explorer II LX workstations, the Knowledge Engineering Environment (KEE) expert system development shell (KEE User's Guide, 1989) and common Lisp (List Processing Language).

Testbed Data Acquisition and Scheduler Interfaces

The testbed data acquisition interface requests pertinent parametric data values from the PDCU controller and asserts new values received into the knowledge base. For incipient fault detection, the data acquisition interface stores the values in a First In First Out (FIFO) table that contains the last 200 values for each analog test point on the testbed. The scheduler writes the load profile to shared memory. A handshaking protocol indicates when the shared memory has been updated with new information. Upon sensing the update, the scheduler interface reads the load profile from memory and updates the knowledge base. Forward chaining fault detection is initiated whenever new values are received in the knowledge base.

User Interface

The APEX system is fully mouse activated for quick and easy operation. A combination of KEE active images and Lisp functions have been developed to provide user graphic screens that display information and menu pick options to the operator. The graphic screens also provide a verification method to assure the system is reasoning correctly. For verification, domain experts set up fault scenarios and review the expert systems diagnosis of the faults and recommended actions.

The operator reviews justification and recommended actions and performs recovery procedures to clear or bypass faults. A longer term goal is to communicate recommended actions as messages to subsystem controllers. The purpose of communicating messages to the subsystem controllers would be to initiate automatic fault correction. Currently, the operator is kept in the fault detection, isolation, and recovery loop as a measure of validation.

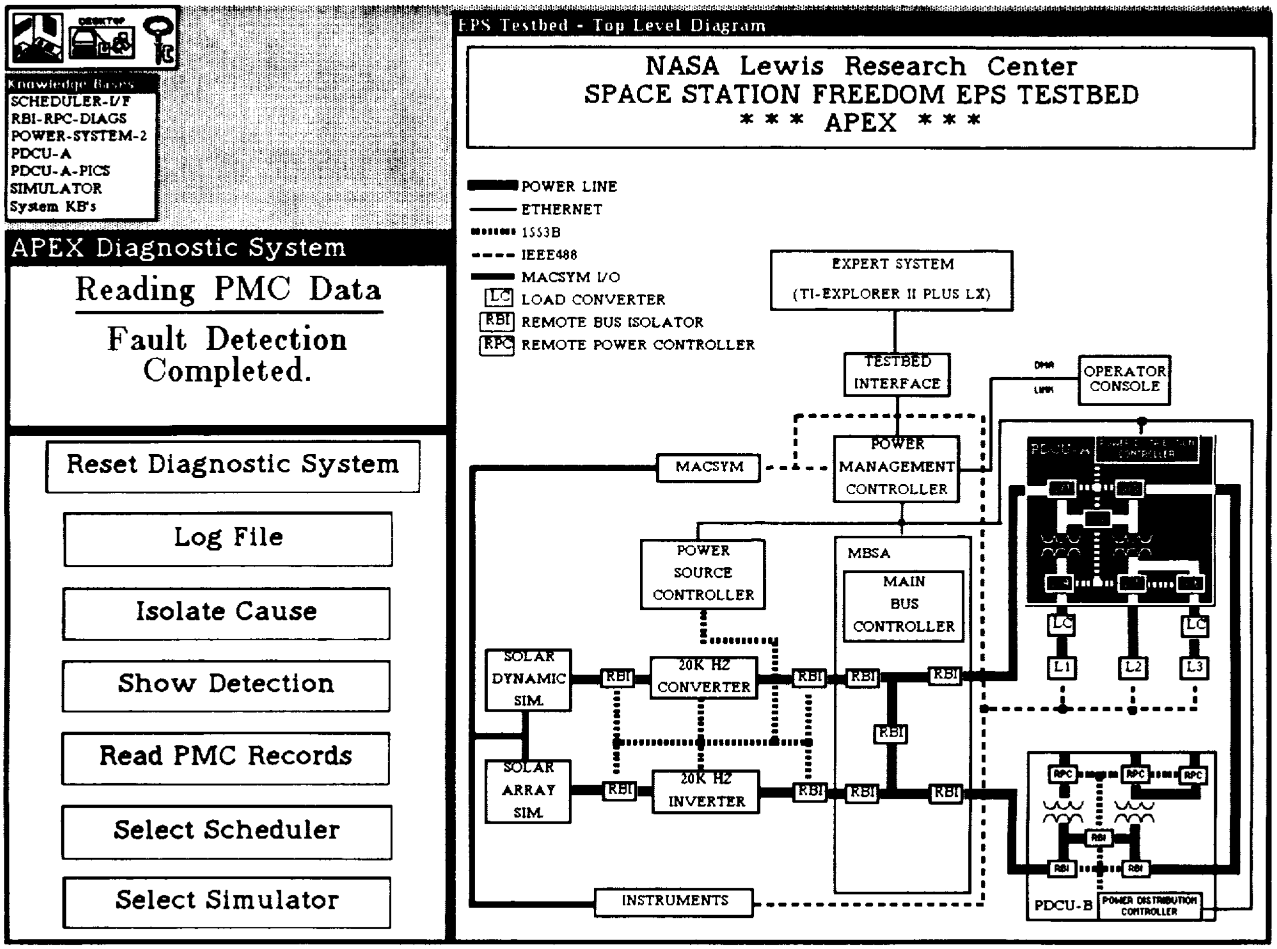
The main user interface screens provide three levels of access to the system. The three levels of access are: (1) a top level block diagram of the SSF/EPS testbed, (2) block diagrams of each subsystem, and (3) subsystem schematic diagrams. Each screen is mouse sensitive for displaying other screens. Visual flashing indications appear on areas of the displays when faults occur. In addition, the schematic display shows the latest voltage, power, phase angle, current, and status values at each device test point. Figures 1, 2, and 3, respectively, show the screens corresponding to the three levels of access.

Three other screens show explanations of fault detection, isolation, and justification. Examples of these three screens are shown in figures 4 to 6.

An example of a recommended action display is shown in figure 7.

There are two screens that correspond to the testbed and scheduler interfaces. An example of the scheduler interface screen appears in figure 8.

There are three screens to display graphical plots of incipient fault data. An example of a ratio plot of measured to expect values for one of the current test points is shown in figure 9. The other two plot types are tolerance and history.



Reading PMC Data

Fault Detection Completed.

FIGURE 1. - TOP LEVEL BLOCK DIAGRAM.

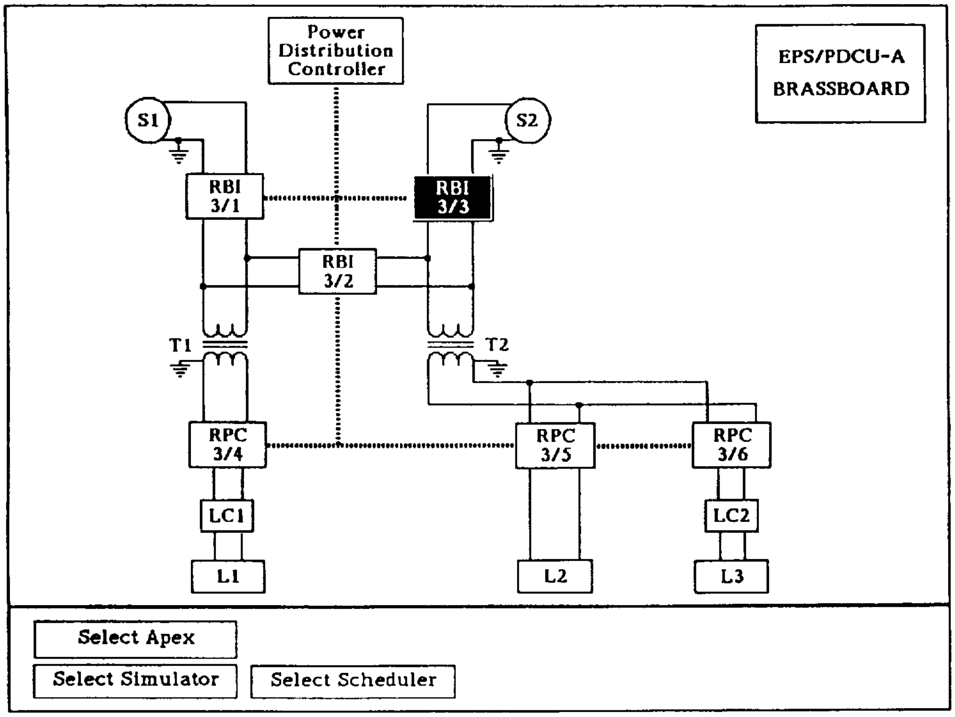
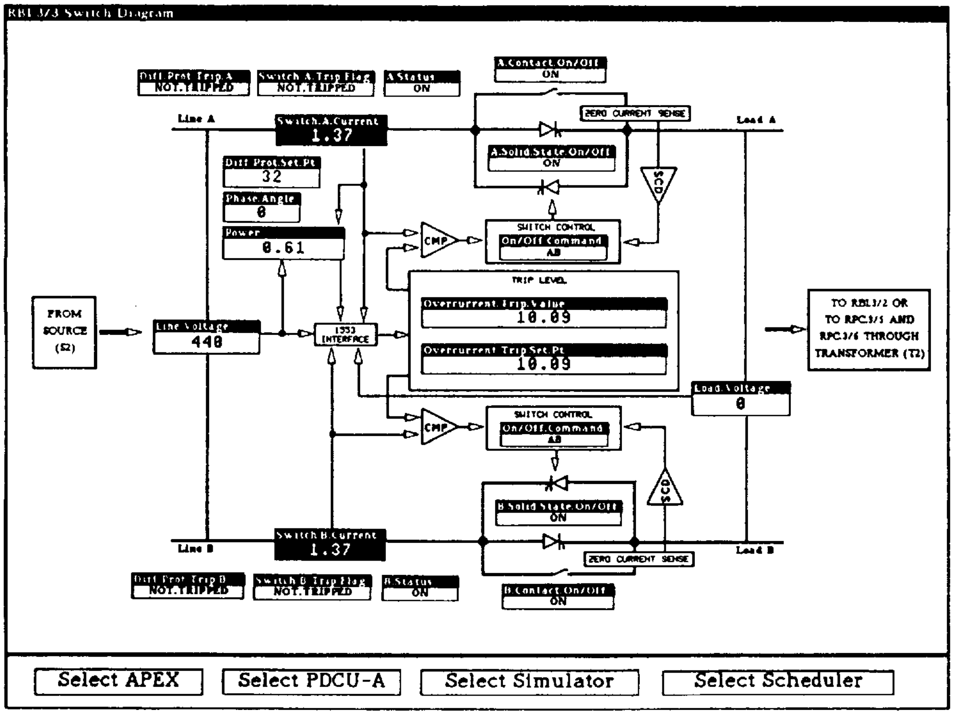
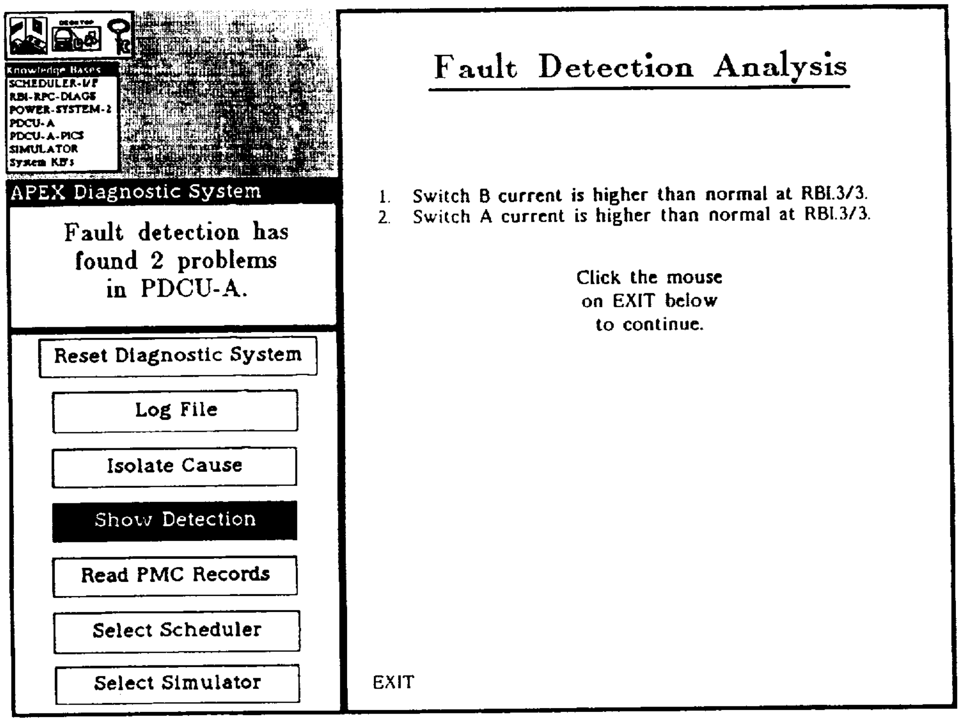


FIGURE 2. - PDCU SUBSYSTEM BLOCK DIAGRAM.


RBI 3/3 Switch Diagram

Diff.Prot.Trip.A: NOT.TRIPPED
Switch.A.Trip.Flag: NOT.TRIPPED
A.Status: ON
A.Contact.On/Off: ON
Switch.A.Current: 1.37
A.Solid.State.On/Off: ON
Diff.Prot.Set.Pt: 32
Phase.Angle: 0
Power: 0.61
On/Off.Command: AB
Overrurrent.Trip.Value: 10.09
Overrurrent.Trip.Set.Pt: 10.09
Line.Voltage: 440
Load.Voltage: 0
On/Off.Command: AB
B.Solid.State.On/Off: ON
Switch.B.Current: 1.37
Diff.Prot.Trip.B: NOT.TRIPPED
Switch.B.Trip.Flag: NOT.TRIPPED
B.Status: ON
B.Contact.On/Off: ON

FIGURE 3. - SCHEMATIC DIAGRAM.



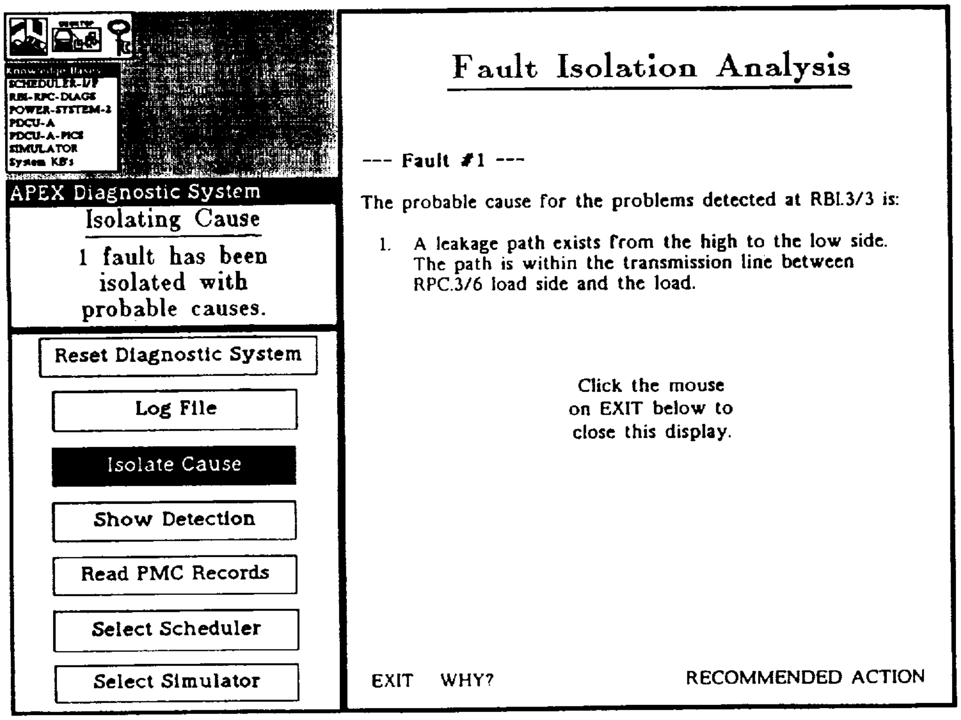
Fault detection has found 2 problems in PDCU-A.

Fault Detection Analysis

1. Switch B current is higher than normal at RBI.3/3.
2. Switch A current is higher than normal at RBI.3/3.

Click the mouse
on EXIT below
to continue.

FIGURE 4. - FAULT DETECTION ANALYSIS SCREEN.



Isolating Cause

1 fault has been isolated with probable causes.

Fault Isolation Analysis

--- Fault #1 ---

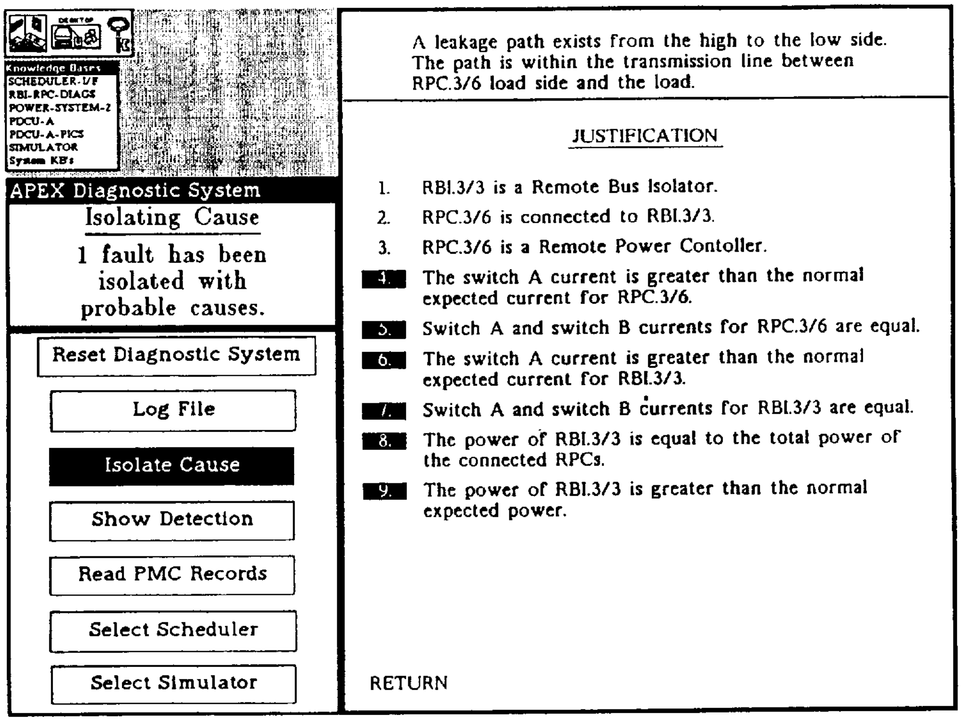
The probable cause for the problems detected at RBI.3/3 is:

1. A leakage path exists from the high to the low side. The path is within the transmission line between RPC.3/6 load side and the load.

Click the mouse
on EXIT below to
close this display.

FIGURE 5. - FAULT ISOLATION ANALYSIS SCREEN.

ORIGINAL PAGE IS
OF POOR QUALITY



APEX Diagnostic System

Isolating Cause

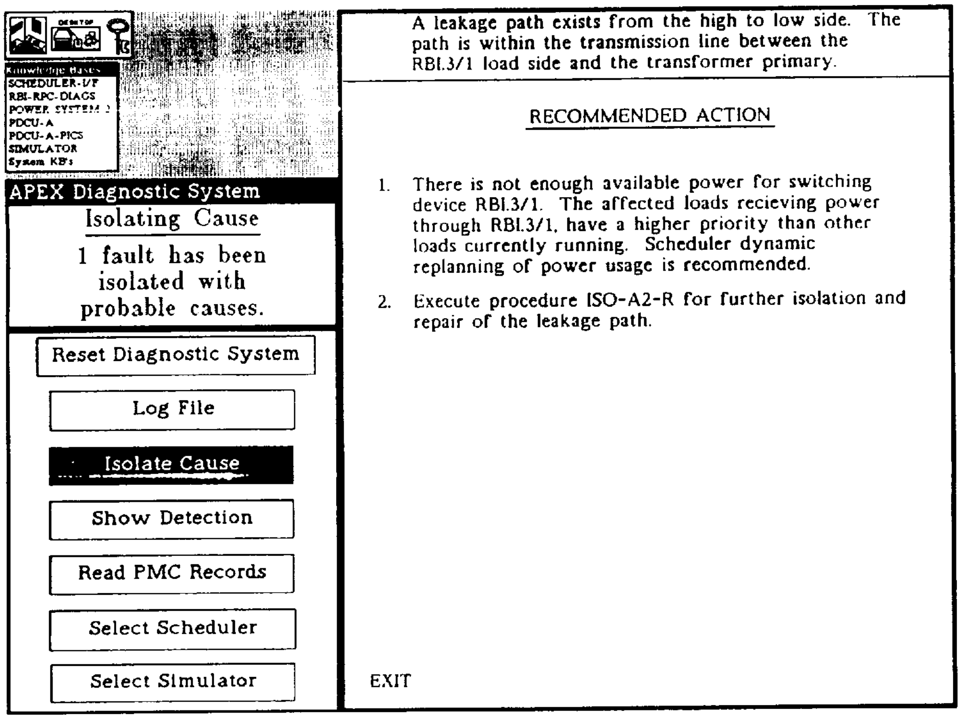
1 fault has been isolated with probable causes.

A leakage path exists from the high to the low side. The path is within the transmission line between RPC.3/6 load side and the load.

JUSTIFICATION

1. RBI.3/3 is a Remote Bus Isolator.
2. RPC.3/6 is connected to RBI.3/3.
3. RPC.3/6 is a Remote Power Contoller.
4. The switch A current is greater than the normal expected current for RPC.3/6.
5. Switch A and switch B currents for RPC.3/6 are equal.
6. The switch A current is greater than the normal expected current for RBI.3/3.
7. Switch A and switch B currents for RBI.3/3 are equal.
8. The power of RBI.3/3 is equal to the total power of the connected RPCs.
9. The power of RBI.3/3 is greater than the normal expected power.

FIGURE 6. - FAULT JUSTIFICATION SCREEN.



APEX Diagnostic System

Isolating Cause

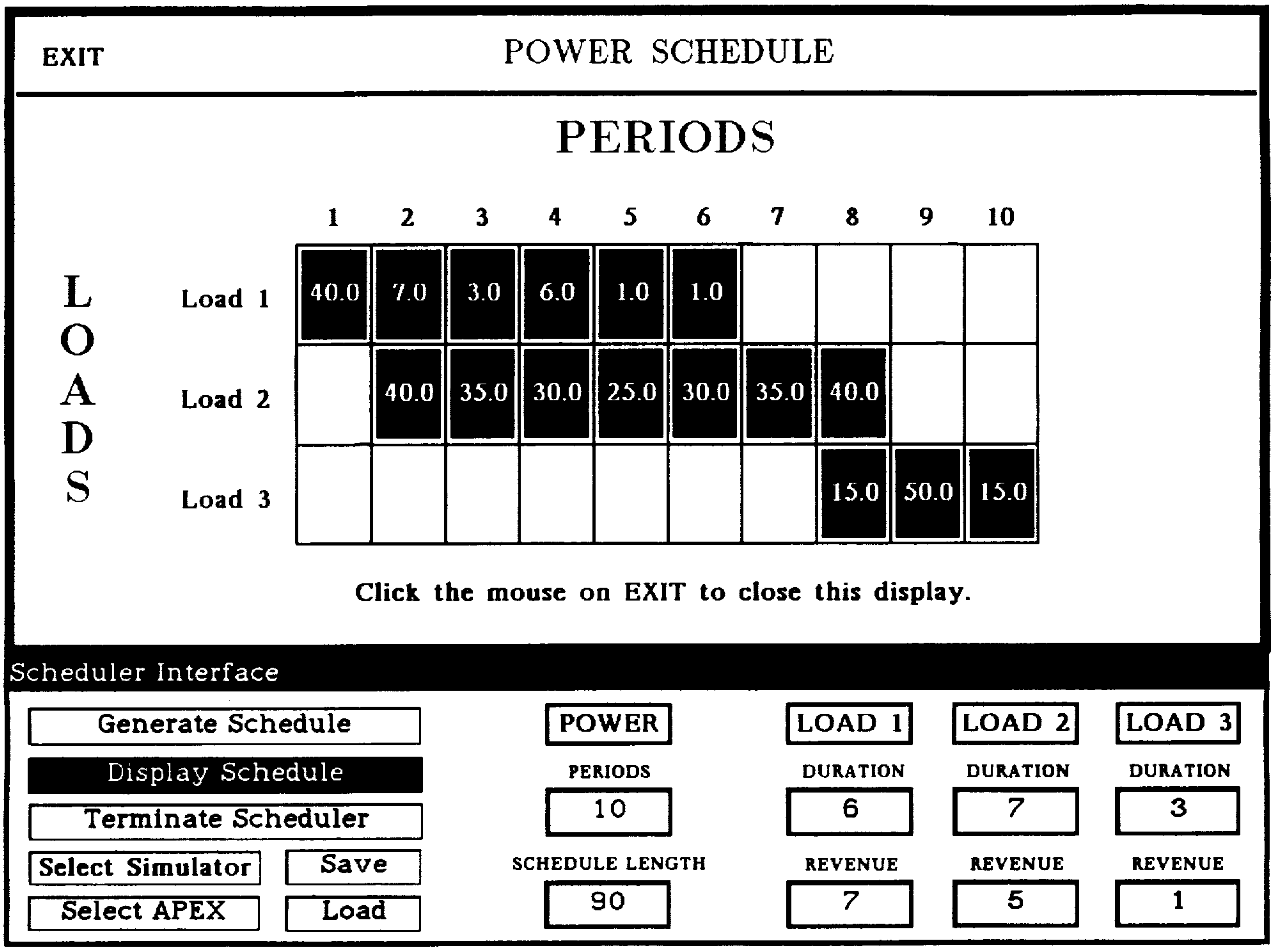
1 fault has been isolated with probable causes.

A leakage path exists from the high to low side. The path is within the transmission line between the RBI.3/1 load side and the transformer primary.

RECOMMENDED ACTION

1. There is not enough available power for switching device RBI.3/1. The affected loads recieving power through RBI.3/1, have a higher priority than other loads currently running. Scheduler dynamic replanning of power usage is recommended.
2. Execute procedure ISO-A2-R for further isolation and repair of the leakage path.

FIGURE 7. - RECOMMENDED ACTION DISPLAY.



POWER SCHEDULE

PERIODS

LOADS

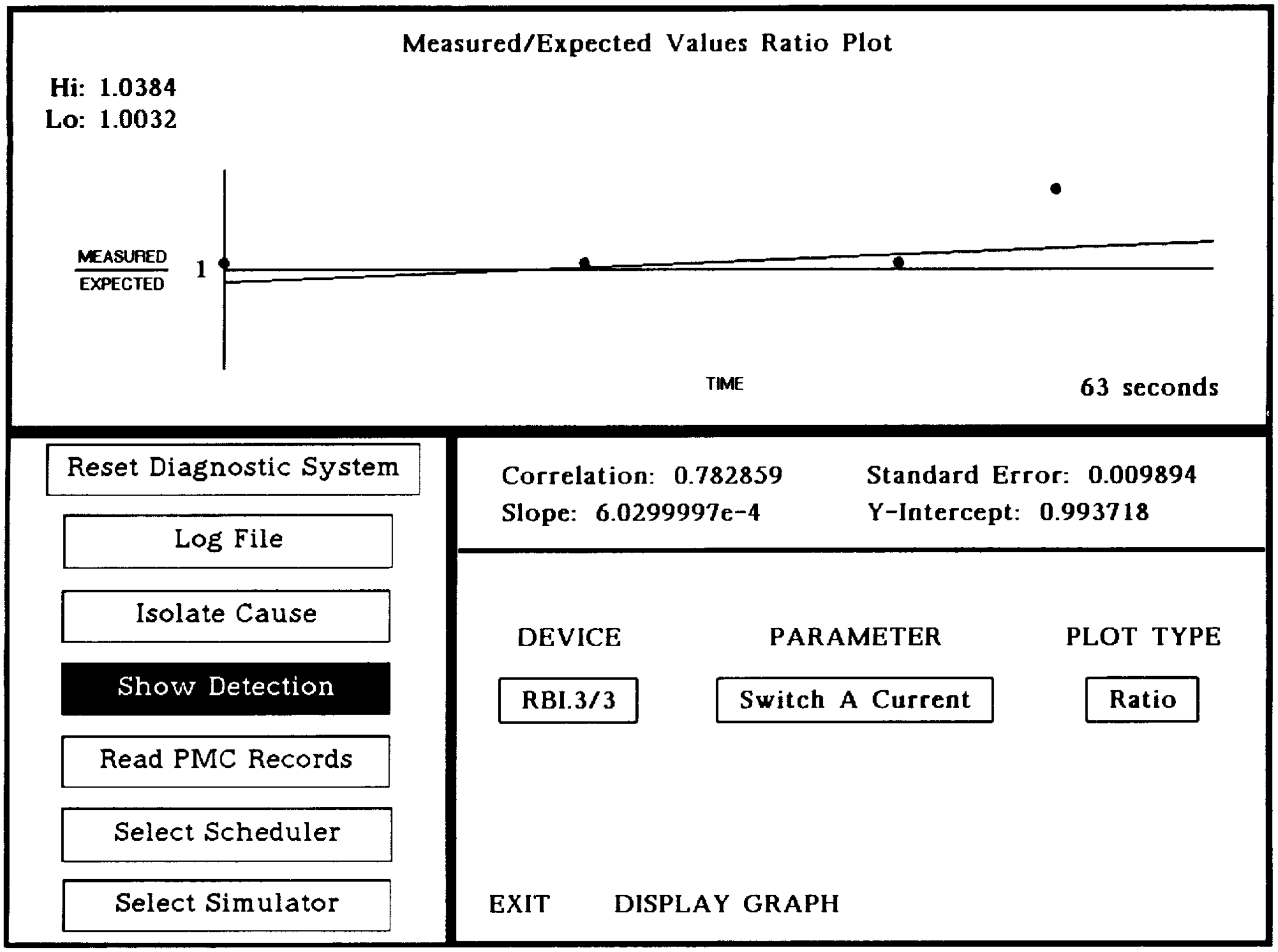
	1	2	3	4	5	6	7	8	9	10
Load 1	40.0	7.0	3.0	6.0	1.0	1.0				
Load 2		40.0	35.0	30.0	25.0	30.0	35.0	40.0		
Load 3								15.0	50.0	15.0

Click the mouse on EXIT to close this display.

Scheduler Interface

POWER	LOAD 1	LOAD 2	LOAD 3
PERIODS	DURATION	DURATION	DURATION
10	6	7	3
SCHEDULE LENGTH	REVENUE	REVENUE	REVENUE
90	7	5	1

FIGURE 8. - SCHEDULER INTERFACE SCREEN.



Measured/Expected Values Ratio Plot

Hi: 1.0384
Lo: 1.0032

Correlation: 0.782859
Standard Error: 0.009894
Slope: 6.0299997e-4
Y-Intercept: 0.993718

DEVICE: RBI.3/3
PARAMETER: Switch A Current
PLOT TYPE: Ratio

FIGURE 9. - INCIPIENT FAULT RATIO PLOT.

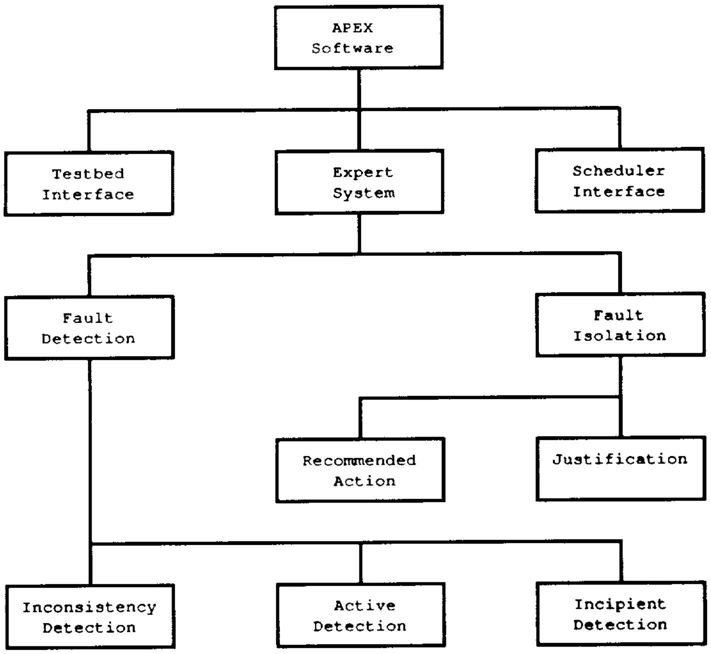


FIGURE 10. - APEX SYSTEM ARCHITECTURE.

SUMMARY AND CONCLUSIONS

APEX has been designed to emulate expert diagnostic fault detection, isolation, and recovery methods. Figure 10 shows the APEX system architecture.

The expert system has fault detection and fault isolation phases. Data values are monitored at each power system test point and compared to expected values derived from a remote power scheduler to detect faults. The testbed interface acquires parametric data values from the testbed. New data values in the knowledge base drive forward chaining for fault detection. Areas of fault detection include inconsistency checks, monitoring for single and multiple active faults and incipient fault analysis. Fault isolation includes justification of probable causes and recommended actions to clear faulty conditions.

The expert system can check more test points, more often than a human operator can, and do so without fatigue. Expert knowledge is continuously available to monitor and diagnose faults in the power system and appropriate recovery procedures are instantly displayed and available to lower level controllers that can command and control the power system. These are valuable benefits for a system such as a space station that will require continuous, long-term health monitoring and autonomous control. Much of the burden placed on human operators can be relieved with the type of expert system technology build into the APEX system.

ACKNOWLEDGMENTS

We would like to acknowledge Eric Bobinsky, Allen Richard, Mark Ringer, Amy Edelman, and Silvia Washington for their work in developing power scheduler software; Mark Ringer for his work in developing the hardware interface between the APEX system and the PDCU subsystem controller; James Dolce and Pamela Mellor for their support, and James Kish, our project manager, for guidance.

REFERENCES

Wright, T; Mackin, M.; and Gantose D.: Development of Ada Language Control Software for the NASA Power Management and Distribution Testbed. NASA Lewis Research Center, Cleveland, OH, 1989.

Truong, L., et al.: Autonomous Power Expert Fault Diagnostic System for Space Station Freedom Electrical Power System Testbed. Third Annual Workshop on Space Operations Automation and Robotics (SOAR 1989), Lyndon B. Johnson Space Center, Gilruth Recreation Center, Houston, TX, July 25-27, 1989 (NASA CP- , to be published).

KEE User's Guide, Version 3.1, Intellicorp, Inc., Mountain View, CA, May 1989.

N90-22307

Spacecraft Command Verification: The AI Solution

Lorraine M. Fesq
Amy Stephan
TRW
Redondo Beach, CA

Brian K. Smith
UCLA
Los Angeles, CA

ABSTRACT

Recently, a knowledge-based approach was used to develop a system called the Command Constraint Checker (CCC) for TRW. CCC was created to automate the process of verifying spacecraft command sequences. To check command files by hand for timing and sequencing errors is a time-consuming and error-prone task. Conventional software solutions were rejected when it was estimated that it would require 36 man-months to build an automated tool to check constraints by conventional methods. Using rule-based representation to model the various timing and sequencing constraints of the spacecraft, CCC was developed and tested in only three months. By applying artificial intelligence techniques, CCC designers were able to demonstrate the viability of AI as a tool to transform difficult problems into easily managed tasks. This paper discusses the design considerations used in developing CCC and examines the potential impact of this system on future satellite programs.

INTRODUCTION

Even after a spacecraft is launched, it continues to receive information from ground stations telling it what actions to perform. This information is in the form of spacecraft commands. These commands are formulated on the ground, transmitted to the spacecraft, and used to instruct the spacecraft to perform actions such as turning instruments on and off, switching relays, or maneuvering the craft into a new orientation. The spacecraft hardware being commanded consists of highly specialized electronics which must be carefully reconfigured. The commands sent to the spacecraft must follow strict guidelines as to their order and timing. These guidelines are known as constraints, and all sequences of commands must be examined to assure that they meet all constraints before they are transmitted to the spacecraft. This pre-checking of command sequences is a very involved and time-consuming task. Performed manually, it could take a week to check one set of commands. Developing a conventional software program to automate this process also would be a difficult and costly task, estimated to take at least 36 man-months. This paper describes an expert system that was developed in only three months to automate the process of checking spacecraft command sequences for constraint violations.

DESCRIPTION OF THE PROBLEM

There are two major types of constraints which must be met within all spacecraft command sequences: *timing* constraints and *ordering* constraints. Consider the following example. A spacecraft is commanded to start recording information onto an on-board tape recorder. This action may involve numerous individual commands, many of them going to the same box on the spacecraft. Each box has a limit of how fast it can receive commands, in much the same manner as humans have a limit of how fast they can absorb sensory input. If commands are sent too fast, the box will not receive all of them, and the spacecraft will not perform the desired action. For example, all commands to the spacecraft Tape Recorder Box must be separated by 50 milliseconds. This is an example of a *timing* constraint.

To accomplish the task of recording data, a number of commands must be sent to the Tape Recorder, and these commands must be in a specific order. An example of an *ordering* constraint would be that the "Tape Recorder On" command must precede the "Tape Recorder Rewind" command, which in turn must precede the "Tape Recorder Record" command.

Spacecraft command sequences can contain hundreds of commands, all of which must meet all timing and ordering constraints. The number of constraints can also be on the order of hundreds. Checking all commands (n) against all constraints (k) would require (n(n-1)/2)*k operations, or order $O(n^2k)$ operations. Checking a medium sized file of 50 commands against 50 constraints could require up to 61,250 operations - an enormous task when performed by hand. An automated system seems an appropriate solution, but developing a conventional software package to perform the checking has been estimated on one spacecraft program to be prohibitive.

The following section describes an AI solution to automating this process, and contrasts the solution to a less efficient, more costly conventional approach.

AI SOLUTION TO THE PROBLEM

The constraints against which commands are checked closely resemble expert system rules. Constraints generally consist of several conditional clauses and an error that will occur if these conditions are met. This format can be easily translated into the type of rules typically used in rule-based production systems. For example:

```
IF    the [tape-recorder rewind] command is received before the
          [tape-recorder on] command
THEN  [constraint X is violated]
```

Commands themselves are symbolic in nature, and generally can be represented as a spacecraft box and an action to be performed on that box. It is more natural to think of a command using its symbolic representation, i.e. [TAPE-RECORDER ON], than its numeric (hexidecimal) representation. Because commands are easily represented as sets of symbols, they can be used as facts in an expert system.

A commercial inference engine could efficiently match facts against rules and note violations, leaving the designer the tasks of writing the rules, converting commands into facts and choosing the appropriate hardware and software tools with which to build the system. An off-the-shelf inference engine seemed the ideal choice for this straightforward production system. By employing such a system, we would have access to efficient unification and database management algorithms, leaving the designer free to concentrate on optimizing the rule and fact representation.

Expert systems seemed ideally suited to solve what had been an unsolvable problem. The set of constraints on satellite commands, although large, is well-documented and would require a domain expert only to explain the highly specialized language in which these constraints are described. Programming an expert system to check constraints would require a straightforward conversion of constraints into rules and a scheme for representing commands as facts. To code this system conventionally would require pages of awkward IF-THEN and CASE statements, performing numeric calculations on data that is inherently symbolic. In addition to the reduction in code size, the expert system approach also promised a significant increase in speed. The inference engine would at most match each command against each constraint clause, an operation of order O(nk) as opposed to order O(n^2k), significantly reducing the time needed check a file of commands.

Within three months, an expert system called the Command Constraint Checker (CCC) was developed (see Figure 1). This system runs on an IBM-compatible PC, which is currently used in NASA Control Centers to write and store command procedures

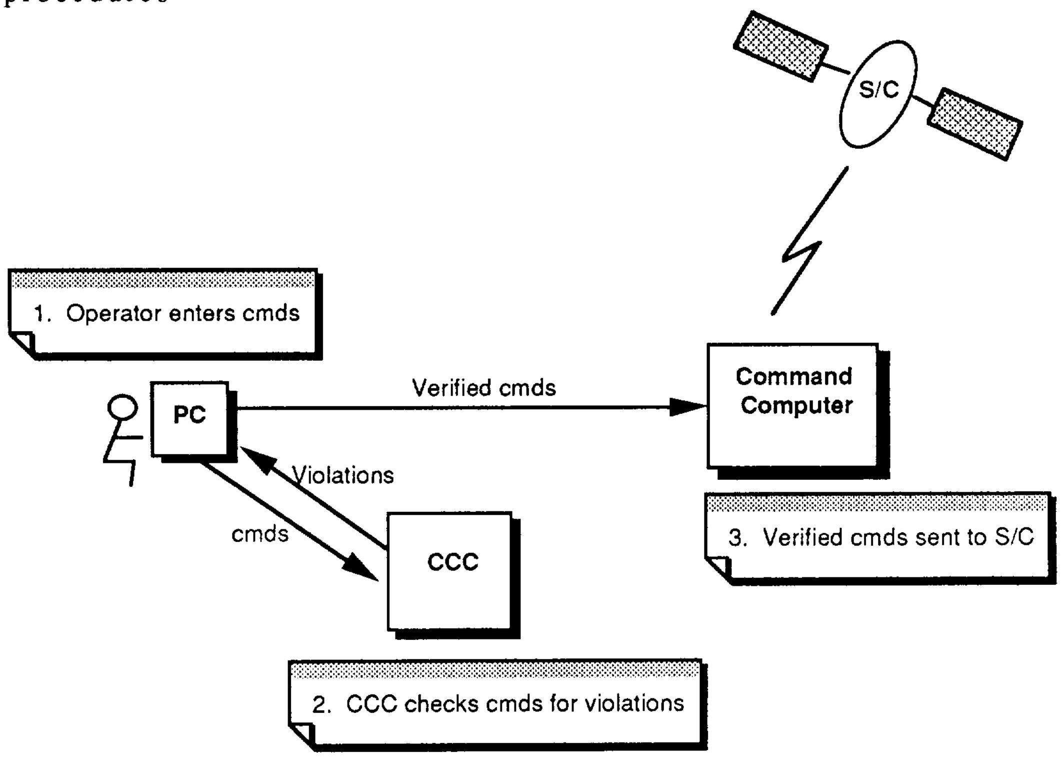


Figure 1. Automated Process of Verifying Spacecraft (S/C) commands

before sending them to a spacecraft. Housing the CCC on the PC allows the program to process actual command files to be sent to a spacecraft, eliminating the need for users to input lengthy command sequences into the CCC. Running the expert system on the PC also allows the CCC to be used on-line; command sequences can be checked immediately before being transmitted.

The Command Constraint Checker consists of a menu-driven user interface, a set of procedures to parse various formats of command files into rules, a rule base, a commercial inference engine and a mechanism for reporting violations to the user. To limit the number of rules in the system, rules are designed to be as general as possible. For example, a constraint might read, "Commands to the tape recorder must be at least 50 ms apart." In the format used in command procedures, however, it is not always easy to recognize which commands control the tape recorder. A set of commands to this instrument might be encoded as follows: CMT1ON, CMT2ON, CMT1OFF, CMT2OFF, CMRWND, CMFF. Using these mnemonics, fifteen rules would be needed to assure that none of the potential tape-recorder command pairs were less than 50 ms apart. If the command designation, TAPE-RECORDER, were included in each command fact, only one rule would be needed to ensure that all tape recorder commands were properly spaced. Using an existing database containing information about all valid spacecraft commands, we were able to abstract command mnemonics into facts amenable to the more general language of the command constraints. As mnemonics are read from a command file, they are matched against the command database and the following information is asserted as part of the command fact:

COMMAND MNEMONIC	As it appeared in the command procedure
HEX CODE	The actual hex representation of the command sent to the spacecraft.
DESTINATION BOX	The box that will receive this command, i.e. tape recorder.
COMMAND DATA	The action to be sent to this box, i.e. REWIND

If a command mnemonic is not found in the database, the command is illegal and this information is included in the user's error report. A routine to optimize the database command mnemonic search is included in the CCC.

The CCC contains parsing routines to convert several types of command procedures into command facts. A menu-driven user interface allows the user to input a command file name, a command database name, the rate at which commands will be sent to the spacecraft and the type of command file to be parsed. The command file formats range from simple lists of commands and WAIT statements, to complicated files containing IF-THEN-ELSE, GOTO and WAIT statements as well as variables. CCC employs the appropriate routines to parse the command file into a list of mnemonics, checks these mnemonics against the database and assigns an absolute time to each command. The CCC bases this absolute time on the user-supplied rate at which commands are being sent to the spacecraft. The first command is given the absolute time of 0, and each successive command is assigned a time based on the uplink rate and the number and duration of WAIT statements in the command file. The line number of the command in the input file also is part of the command fact. If the command fact causes a violation, this line number is included in the user's error report, allowing her to easily edit the original command file. A command fact with all its fields might look like this:

[CMT1OFF	TAPE-RECORDER	OFF	133A077	.128	5]
mnemonic	destination	data	hex code	absolute time	line

While CCC automatically converts commands into facts, the designer must represent constraints as rules. The task of maintaining the rule base is simplified by several factors. All constraints on commands for a given satellite typically are well-documented in that satellite's mission operations handbook. The page in the handbook on which a constraint appears is included in that constraint's documentation. This allows the engineer in charge of maintaining the system to easily delete a rule if the constraint it represents is later deemed unnecessary. Conversely, the close correlation between IF-THEN rules and the language in the constraint descriptions will allow an engineer with little or no background in expert systems to add new constraints to the rule base.

The CCC required a compact forward-chaining inference engine capable of interfacing with a conventional language and running on a PC. CLIPS, a NASA-built expert system shell written in C meets all of these needs. Since CLIPS rules can be run from within a C program, we were able to embed the expert system in a conventional C program. This allowed us to perform procedural tasks, such as the user-interface and file operations, in C and actually identify the constraint violations using CLIPS (Figure 2). After all inferences have been made, control returns to the C program which records the CLIPS violation facts in a readable file for the user.

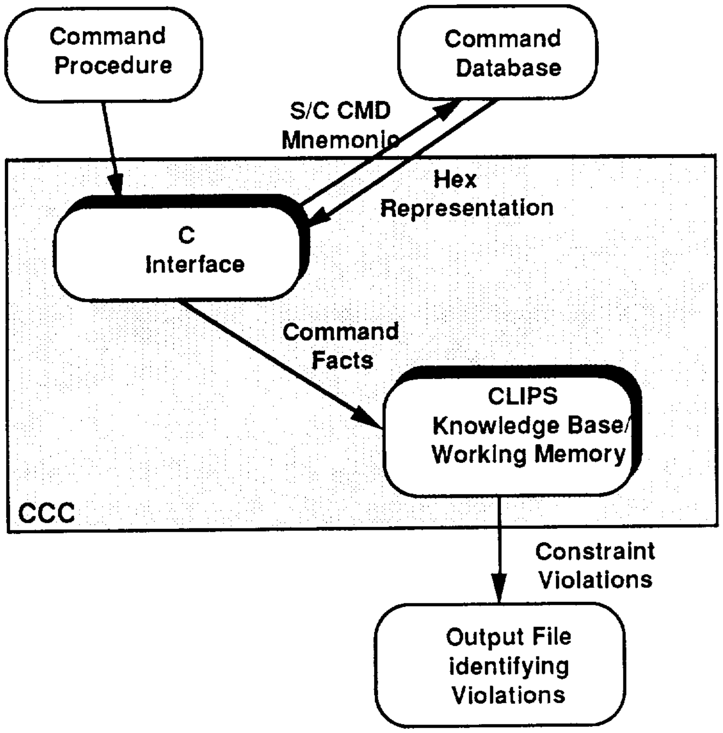


Figure 2. CCC Block Diagram

STATUS

After three months of coding, a working version of the CCC is ready to be delivered to its users. The program solicits user input through a menu-driven interface, parses three formats of command files, checks these commands against a command database and asserts them as command facts. It then runs its rule base of 70 rules against the facts, and provides the user with a readable output file listing the nature of all violations and warnings, including the line number in the original command file of each flagged command. This information allows the user to edit her command procedure and re-check it with CCC before sending the commands to the spacecraft.

A rule base for a specific spacecraft, NASA's Gamma Ray Observatory (GRO), was developed to test the CCC. The 70 rules now in the CCC database cover about 90 percent of all GRO command constraints, including the constraints on those commands most often sent. A detailed user's guide has been developed, explaining how the CCC may be adapted to a new spacecraft, as well as how it may be modified to accommodate new constraints that arise during the life of the spacecraft. Potential users have been given the opportunity to experiment with CLIPS and gain familiarity with its rule structure so that they can maintain the rule base with little aid from the designers. The system is able to check a typical file of 50 commands in five minutes. This is a considerable savings over the hours that previously were spent checking command sequences manually.

No major problems were encountered while developing CCC. The choice of languages and the hardware platform proved easy to use and adequate for our needs. Excellent cooperation from domain experts and potential users sped the development of this system, allowing us to complete the project on schedule and within budget.

EFFECTIVENESS OF IMPLEMENTATION

Response from users has been enthusiastic. In several beta tests and demonstrations, users have been impressed by the performance and usability of CCC. Engineers who plan to apply this tool are happy to be free of the tedious chore of checking command sequences by hand. Management also is pleased with the results of the project, which produced an automated tool for less than 10 percent of the estimated cost of building such a system using conventional methods.

CCC is a classic example of how examining a software problem from an AI perspective can change the nature of the problem. By observing the symbolic nature of the commands and constraints, the CCC designers were able to transform a difficult conventional problem into a relatively straightforward expert system task. The application of artificial intelligence techniques to this problem produced a useful tool that will save many wasted hours, thousands of dollars and potentially the life of a spacecraft.

REFERENCES

1. Fesq, L., & Stephan, A. (February 1989). Advances in Spacecraft Autonomy Using Artificial Intelligence Techniques. *12 Annual AAS Guidance and Control Conference*, Keystone, Colorado.

1. Giarratano, Joseph C. (1988). *CLIPS USER's Guide*. Artificial Intelligence Section, Lyndon B. Johnson Space Center.

2. Knuth, Donald E. (1973). *The Art of Computer Programming*, Vol. 1. Addison-Wesley Publishing Co., Reading, Massachusetts.

3. Nilsson, Nils J. (1980). *Principles of Artificial Intelligence*. Morgan Kaufman Publishers, Inc., Los Altos, CA.

4. Waterman, Donald A. (1986). *A Guide to Expert Systems*. Addison-Wesley Publishing Co., Reading, Massachusetts.

Analyzing Spacecraft Configurations Through Specialization and Default Reasoning

Matthew R. Barry
Carlyle M. Lowe

Rockwell Space Operations Company
600 Gemini Ave., R20A-4
Houston, TX 77058

19 January 1990

Abstract

For an "intelligent" system to describe a real-world situation using as few statements as possible, it is necessary to make inferences based on observed data and to incorporate general knowledge of the reasoning domain into the description. These reasoning processes must reduce several levels of specific descriptions into only those few that most precisely describe the situation. Moreover, the system must be able to generate descriptions in the absence of data, as instructed by certain rules of inference. The deductions applied by the system, then, generate a high-level description from the low-level evidence provided by the real and default data sources.

We describe an implementation of these ideas in a real-world situation. The application concerns evaluation of Space Shuttle electromechanical system configurations by console operators in the Mission Control Center. A production system provides the reasoning mechanism through which the default assignments and specializations occur. We provide examples within

this domain for each type of inference, and discuss the suitability of each toward achieving our goal of describing a situation in the fewest statements possible. Finally, we suggest several enhancements that will further increase the "intelligence" of similar spacecraft monitoring applications.

1 INTRODUCTION

This paper addresses the application of default reasoning and specialization techniques toward problems involving pattern classification. A collection of discrete sensor values from a real-time telemetry stream are integrated with certain knowledge about the "world" these sensors represent in order to synthesize an understanding of a situation. By delimiting various intersensor relationships and applying them to subsets of the sensor space, *specializations* of certain situations are achieved. These specializations reduce the number of propositions in the world while maintaining a sort of "semantic equivalence."

In some cases certain sensor values may be missing. This absence of information may be due to some problem, or it may be evidence of a feasible situation in which the *lack* of information is information in itself. For these situations it is reasonable to allow a *default* value for a sensor. This default value may be specified *a priori* or somehow derived from the current context.

1.1 Specialization

Specialization is a fundamental reasoning process employed in *configuration analysis*. By configuration analysis we mean the process of evaluating all of the sensor values within a given context. This can be accomplished by building an evaluation step-by-step from the lowest (least encompassing) statements to the highest (most encompassing) statements. In essence, this process classifies patterns of labelled binary samples into prespecified groups, each of which becomes a sample on its own. The hierarchy of groups represents the specialization of several samples into one *equivalent* sample.

For instance, if

$$\{\phi_1, \ldots, \phi_n\} = P$$

where ϕ_i are labelled samples from context Φ, then

$$\bigwedge_{\phi_i \in P} \Leftrightarrow \psi_1$$

represents the specialization of the statements ϕ_i into the statement ψ_1. Furthermore, if

$$\{\psi_1\} \cup \{\phi_{n+1}, \ldots, \phi_{n+m}\} = Q$$

then we might apply the successor specialization

$$\bigwedge_{\phi_i \in Q} \Leftrightarrow \psi_2$$

and so on. For efficiency in our production system implementation (described below), we restrict each sample to one specialization by removing it from the context (database) as it is consumed by the new statement. Considering the last example, we apply

$$\begin{aligned} \Phi &= \overline{Q}(in\Phi) \\ \Phi &= \Phi \cup \{\psi_2\} \end{aligned}$$

after the specialization.

1.2 Default Reasoning

The problem area we consider in this paper belongs to the group of problems in Artificial Intelligence research labelled *common sense reasoning.* In order to draw conclusions based upon certain conditions in an "intelligent" manner, there must somehow be a higher level of practical information that represents what we might ascribe to a human as "common sense." For example, we might reason "if b is a bird, and we have no reason to believe that b cannot fly, then conclude that it can."

Research efforts related to solving these problems have centered around extending classical mathematical logics to account for implicit information in the database. Typically, this is done by making assumptions about missing information by providing default values. In some cases, providing default values is in itself another problem that must be handled in the reasoning system. Etherington [Etherington 1988] provides a summary of current techniques for handling incomplete information.

1.2.1 The Closed-World Assumption

In an attempt to restrict the reasoning assumptions to information that is available, the *Closed-World Assumption* (CWA) has been developed [Reiter 1978]. The CWA is the assumption of complete knowledge about which positive facts are true in the world. Under the CWA, it is not necessary to explicitly represent negative information. Negative facts may be inferred from the absence of the same positive fact. The CWA corresponds to the knowledge base:

$$\text{if } KB \nvdash P \text{ then infer } \neg P,$$

which states that if the proposition P cannot be derived from the knowledge base KB, then it is reasonable to assume that P is false.

1.2.2 Default Logic

Traditional logics do not possess means for considering the absence of knowledge. Research has considered two sorts of information types whose implementation can extend the capabilities of traditional logics to cover this shortcoming[1]. In the *positive information* category, one assumes that rele-

[1] A formal introduction to default logic may be found in [Besnard 1989].

vant information is known, therefore anything that is not known must be false. In the *default information* category, one has default values available to fill gaps in the absence of specific evidence. It is this *default information* category that describes the reasoning process embodied by our classifier.

A *default logic* may be constructed from a standard *first-order logic* by permitting addition of new inference rules [Reiter 1980,Reiter and Criscuolo 1981]. These new rules allow *known* and *unknown* premises, making possible conclusions based on missing information. A *default theory*, Δ, is an ordered-pair (D, W) consisting of a set of *defaults*, D, and a set of *first-order formulae*, W. The fundamental statements in Δ are *defaults*, defined by the expression:

$$\frac{\alpha(\overline{x}):\beta_1(\overline{x})\ldots\beta_m(\overline{x})}{\gamma(\overline{x})}$$

where $\alpha(\overline{x})$, $\beta_i(\overline{x})$, and $\gamma(\overline{x})$ are formulae whose free variables are contained in $\overline{x} = x_1, \ldots, x_n$. This expression states that if certain *prerequisites* α are believed, and it is consistent to believe that certain *justifications* β are true, then it is reasonable to sanction the *consequent* γ [Etherington 1988]. If $\beta(\overline{x}) = \gamma(\overline{x})$, then the default is *normal.* If $\beta(\overline{x}) = \gamma(\overline{x}) \wedge \omega(\overline{x})$, for some $\omega(\overline{x})$, then the default is *semi-normal.*

This capability to withdraw a previous assumption and reconstruct a new set of conclusions is known as *nonmonotonic reasoning* [Ginsberg 1987].

2 Application

The application we present involves the operational evaluation of Space Shuttle electromechanical component configurations by flight controllers in the Mission Control Center (MCC). Specifically, specialization and default reasoning techniques have been applied to one of the tasks involved in monitoring two Shuttle propulsion subsystems: the *Orbital Maneuvering System* (OMS) and the *Reaction Control System* (RCS).

2.1 Overview

To operate the OMS and RCS, Shuttle astronauts manipulate a collection of switches controlling valves that direct the fluid flows throughout a plumbing network. Many of these switches control two valves simultaneously: an oxidizer system valve and the corresponding fuel system valve. Position indicators within the valves and switches provide insight into their mechanical position.

Flight controllers in the MCC help the astronauts to manage these systems by monitoring the on-board configuration. The information available to the flight controllers is more complete than the information available to the astronauts Valve and switch positions appear to the flight controllers as binary values noting presence of (or lack of) an open indication, close indication, or both. A set of 16-bit *configuration words* relay all of the available measurements to the flight controllers.

The MCC computers help the flight controllers to monitor the on-board valve and switch configuration by executing a program that compares *actual* and *expected* configurations. Since only some of the bits in a given configuration word apply to the systems of interest, the comparison procedure includes a set of masking words. When the bit patterns that are not filtered by the mask disagree, the program indicates a problem by displaying a certain status character next to that word. Since the contents of those words are displayed in hexadecimal notation, the operators are made aware of a discrepancy condition through this status character, but are not informed of the specific discrepancy. Furthermore, several discrepancies may occur in the same word.

The process of manually deciphering the hexadecimal information is time consuming and prone to error, so we use a computer program to decode any word of interest. This program displays English descriptions of the indications corresponding to those bits that do not match the expected bit pattern. It is up to the operator, however, to remember the patterns from each individual decoding, and to construct a complete signature interpretation from several hexadecimal words simultaneously.

2.2 Reducing Information

The classifier we describe was developed to perform this decoding and *signature construction* task through belief specialization and default reasoning. The decoding program was extended to isolate each bit in the configuration words and to generate a *statement* for a database describing the observed and expected indications. The classifier then attempts to generate a state description for these indications. The state descriptions offer an explanation in high-level, intuitive, terminology. For example, instead of being offered the four statements

```
Open(ox-valve,manifold-1)
Open(fu-valve,manifold-1)
¬Closed(ox-valve,manifold-1)
¬Closed(fu-valve,manifold-1)
```

the flight controller is informed

```
Open(valves,manifold-1)
```

due to the application of a typical specialization rule

```
Open(ox-valve,x) ∧
Open(fu-valve,x) ∧
¬Closed(ox-valve,x) ∧
-Closed(fu-valve,x) ⇐
Open(valves,x)
```

where `x` is bound to `manifold-1`. Better still, if the database includes the statements

```
Open(valves,manifold-1)
Open(valves,manifold-2)
Open(valves,manifold-3)
Open(valves,manifold-4)
Open(valves,manifold-5)
```

then the best description is

```
Open(valves,all-manifolds)
```

from the specialization

```
Open(valves,manifold-1) ∧
Open(valves,manifold-2) ∧
Open(valves,manifold-3) ∧
Open(valves,manifold-4) ∧
Open(valves,manifold-5) ⇔
Open(valves,all-manifolds)
```

Carrying on to "meta-level" statements regarding a "configuration of configurations," one might make the specialization of the statements

```
Open(valves,all-manifolds)
Open(valves,rcs-regulators)
Open(valves,loms-crossfeed)
Open(valves,all-prop-tanks)
On(heaters,thrusters)
Off(heaters,pods)
⋮
```

resolve to the implicit description

```
Configuration( Prelaunch )
```

2.3 Missing Information

One important consideration in the problem is that *lack of evidence regarding a position indication is important information.* That is, missing information may imply a certain position indication. For the OMS and RCS, this is the case with the switch positions: lack of an OPEN or CLOSED indication means that the switch is assumed to be in the GPC (General Purpose Computer) position for computer-controller valve operation. This corresponds to the semi-normal default rule (without prerequisites)

$$\frac{:\mathtt{GPC(s)} \wedge \neg\mathtt{Open(s)} \wedge \neg\mathtt{Closed(s)}}{\mathtt{GPC(s)}}$$

for switch s.

Missing information is also important in valve positions. Many valves lack instrumentation of the CLOSED position, so if the OPEN indication is not present, then one must assume that the valve is closed. Similar to the switch position default, this corresponds to the semi-normal default rule

$$\frac{:\mathtt{Closed(v)} \wedge \neg\mathtt{Open(v)}}{\mathtt{Closed(v)}}$$

for valve v.

3 Implementation

The sort of reasoning process described above can be implemented through the use of a commercial production system. Statements providing a *specialization of beliefs* conveniently can be represented as conventional production rules. The left-hand side of the rule consists of one or more statements which, when considered together, imply a more specialized statement having equivalent meaning. The right-hand side of the rule asserts the consequent statement and retracts all of the prerequisites that were held true in order to fire the rule. This process decreases the total number of statements in the database, while maintaining equivalent knowledge within the reasoning world. The system can retract its own conclusions (and assumptions) later in the deduction process, thereby exhibiting nonmonotonic reasoning.

3.1 Design

The application we describe uses a combination of procedural and declarative programming techniques. NASA's C Language Integrated Production System (CLIPS) provides rule processing capabilities. A host program, written in C, acquires the necessary data and applies a valuation algorithm to generate statements (facts) for the database. This algorithm assigns to each positive component position indication a description of the component, a description of the position indication (e.g. **Open**, **Closed**, **On**, or **Off**), and a qualifier as to whether that position belongs to the *actual* or *expected* configuration. When all necessary statements have been generated, the production system evaluates them and builds the state description with the given inference rules. The contents of the database after all inferences have been performed (i.e. when no more rules fire) represents the conflict set between the actual and expected configurations. The host program translates this set of statements into English sentences for display to the operator.

Since the independence of valve or switch state indications is not guaranteed by the physical system, so this independence is not required by our production system. That is to say, though the valves are intended to reside in either the opened or closed states, the indications may not provide conclusive evidence and perhaps no default assumptions are available. For these situations none of the statements that consider the guilty valve will be applied, thereby leaving the lowest level samples in the database. This is a desirable characteristic of the program, causing it to provide all of the evidence that was not reduced through the inference process. Moreover, facts are held based on observed world states rather than assumed states[2].

In order to reason about defaults one must be able to decide when information is missing. Our application uses the CLIPS *not* operation for this purpose. This operation evaluates to TRUE if a match is *not available* for the pattern, thus allowing us to determine that default-overriding evidence is not present in the database. If the default indication is the only one

[2] There remains the underlying assumption, however, that the observed state represents the actual state.

available for a particular sensor, then the value provided as the default value for that sensor becomes the value of the missing pattern. If any evidence other than the default value is available, that evidence is used in the classification process. These *default processing rules* fire first so as to build all of the lowest-level indications before starting specializations. A typical default rule looks like this:

```
(defrule expect-switch-defaults
 (default ?dom ?item
      ?d&sp-op|sp-cl|sp-gp)
 (not (actual ?dom ?item sp-op))
 (not (actual ?dom ?item sp-cl))
 (not (actual ?dom ?item sp-dm))
 (not (actual ?dom ?item sp-gp))
=>
 (assert (actual ?dom ?item ?d))
)
```

This rule extracts a default indication from the default table, specifying that it handles only switches by restricting the pattern match to one of the three reasonable switch values (the value of *dilemma* (**sp-dm**), though a possible observed state, is not a reasonable default value). It then proceeds to search for an overriding indication by looking for all possible switch values in the **actual** indications. If a match is found, then an **actual** indication is present and the rule fails. If no match is found then the default value is appropriate, so the rule fires, asserting the default value as the **actual** value.

Most of the production rules in our application represent the *specialization rules*. These rules assemble collections of patterns into a more specialized pattern implying the same information. The right-hand side of the rule retracts the premises and asserts the conclusion. Each of these rules works for either of the two comparison states. Recalling the manifold example provided earlier we demonstrate a specialization rule as shown below. This rule collects all five of the named manifolds for an arbitrary domain **dom** and either specialization mode (**actual** or **expect**). Providing the switch and valve positions (**?s** and **?v**) for each manifold are the same, the rule asserts the special conclusion **?dom manifolds**. Prior to the special assertion, however, the rule retracts the prerequisites from the database[3].

```
(defrule specialize-group-manifolds
 ?m1 <- (?mode&actual|expect
        ?dom manifold-1 ?s ?v)
 ?m2 <- (?mode
        ?dom manifold-2 ?s ?v)
 ?m3 <- (?mode
        ?dom manifold-3 ?s ?v)
 ?m4 <- (?mode
        ?dom manifold-4 ?s ?v)
 ?m5 <- (?mode
        ?dom manifold-5 ?s ?v)
=>
 (retract ?m1 ?m2 ?m3 ?m4 ?m5)
 (assert(?mode
        ?dom manifolds ?s ?v))
)
```

4 Extensions

Though the techniques we have employed constitute a powerful application, there are a variety of enhancements that can be made to the reasoning process. We outline a few of them here.

4.1 Temporal Reasoning

Comparing an *actual* signature with an *expected* signature can sometimes be interpreted as a matter of temporal persistence. If we can make assumptions

[3] The retraction is performed before the assertion to minimize the complexity in driving new patterns through the network.

about the dynamic behavior of the measured system, then we can draw from knowledge of the *expected* state to help make assumptions about the *actual* state.

One can imagine running a configuration evaluator continuously (ours runs only upon demand), focusing only on those indications that change in the signature. An interesting enhancement therefore might be in *predicting the next signature* by incorporating knowledge of procedures and time [Georgeff 1987].

4.2 Analog Reasoning

Though the information provided as input to the classifier currently is discrete (binary), analog information may also be important in describing a configuration. For example, some valves may not have discrete position indications, but rather "percentage open" indications. There may be guidelines for interpreting "percentage flow" through these valves that could be implemented as rules with thresholds on their left-hand sides. If a valve is indicating 2% open, for example, the interpretation will probably lead to considering this valve closed.

Analog interpretations may also be used to reason about system measurements that are not strictly part of the "configuration." We might include considerations for thermodynamic measurements in our evaluation, building flow hierarchies, limit violation detectors, or determining "degrees of wellness" for analog components.

4.3 Evidential Reasoning

A variety of problems may be introduced into the classification process by supplying nonrepresentative signatures as input. There are many orbiter component failures that will cause an invalid signature to be relayed to Mission Control. For example, failure of a computer, demultiplexer, signal conditioner or transducer will cause all of the telemetry measurements associated with that components to be incorrect, without affecting operation of the measured device. These conditions are detectable, however, and can be provided as input to the classifier. When the classifier is made aware an instrumentation component failure, and it "knows" the measurements derived from that component, then it can take this invalid information into account when performing the classification.

Sometimes the instrumentation failure may not be known before a classification process begins. In these cases it might be useful to refer to *sub-signatures* that one can map onto the actual signature, measuring the degree to which each body of evidence supports the indicated signature. The heuristics for interpreting competing signatures will likely involve *evidential reasoning* [Lowrance 1986].

5 Summary

The motivation behind this project has been to desire to demonstrate the capabilities of applied default reasoning and specialization as realized through a typical production system. We described a system that implements these reasoning paradigms in a real-time telemetry monitoring application. This application performs a complete task, relieving flight controllers from this duty and allowing them to address their attention to other activities. Due to its declarative construction, the system is able to accommodate changes in the "world" without restructuring the inference process. Most importantly, the system is able to perform a mundane task frequently, consistently, and inexpensively, while producing expert-level results.

We also described several enhancements that seem to be logical extensions to the current system. These extensions will be investigated in the near future.

References

[Besnard 1989] Besnard, *An Introduction to Default Logic*, Springer-Verlag, Berlin, 1989.

[Etherington 1988] Etherington, *Reasoning with Incomplete Information*, Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1988.

[Georgeff 1987] Georgeff and Lansky, "Procedural Knowledge," SRI International Technical Note 411, Menlo Park, CA, 1987.

[Ginsberg 1987] Ginsberg, *Readings in Nonmonotonic Reasoning*, Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1987.

[Lowrance 1986] Lowrance, Garvey and Strat, "A Framework for Evidential- Reasoning Systems," in *Proceedings of the Sixth National Conference on Artificial Intelligence*, 1986.

[Reiter 1978] Reiter, "On Closed-World Data Bases," in *Logic and Data Bases*, Gallaire and Minker (eds.), Plenum Press, New York, 1978.

[Reiter 1980] Reiter, "A Logic for Default Reasoning," *Artificial Intelligence 13*, North-Holland, 1980.

[Reiter and Criscuolo 1981] Reiter and Criscuolo, "On Interacting Defaults," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, 1981.

SIMULATION-BASED REASONING ABOUT THE PHYSICAL PROPAGATION OF FAULT EFFECTS*

Stefan Feyock
Dalu Li

Department of Computer Science
College of William & Mary
Williamsburg, VA 23185

*Supported by NASA/Langley Grant NCC-1-122

Abstract - The research described in this paper deals with the effects of faults on complex physical systems, with particular emphasis on aircraft and spacecraft systems. Given that a malfunction has occurred and been diagnosed, the goal is to determine how that fault will propagate to other subsystems, and what the effects will be on vehicle functionality. In particular, we describe the use of qualitative spatial simulation to determine the physical propagation of fault effects in three-dimensional space.

INTRODUCTION

The work described in this paper was performed in conjunction with the fault management research under way at the Vehicle Operations Research Branch of NASA/Langley Research Center. The goal of this research is to produce software that can serve as an in-flight pilot's aid to assist the flight crew when feasible. In particular, artificial intelligence (AI) techniques are being used to construct systems that will assist flight crews in dealing with in-flight malfunctions.

Any system malfunction raises three categories of questions: what has gone wrong (diagnosis), how will the system be affected (prognosis), and what should be done about it (recovery planning). Fault diagnosis is handled by an array of techniques including traditional rule-based systems, model-based monitoring (MONITAUR [Schutte]), and model-based reasoning from first principles (DRAPHYS [Abbott]). The research described in this paper is concerned chiefly with the prognosis of fault propagation, and takes as input the diagnoses produced by the DRAPHYS system. The physical propagation of fault effects is then simulated to determine possible effects on the air/spacecraft. It is also the case, however, that similar techniques can be run off-line to help construct the physical dependency net used by DRAPHYS. Since DRAPHYS plays a major role in this research, we begin by giving a brief description of this system.

THE *DRAPHYS* FAULT DIAGNOSIS SYSTEM

DRAPHYS reads in a database describing a set of components, predicates indicating which components are sensors and with which non-sensor components the sensors are associated, and predicates describing functional and physical dependencies among components. For example, the predicate *Sensor(N2B(CompressorB))* indicates that *N2B* is a sensor associated with jet turbine component *CompressorB*.

A component Y is deemed to be functionally dependent on another component X if a malfunction in X can affect the functioning of Y. A malfunction in CompressorB, for example, will affect the operation of CombustorB. Clearly any sensor associated with component X is functionally dependent on X. DRAPHYS uses such functional dependency information in its model of the physical system.

The other kind of dependency information utilized by DRAPHYS is physical dependency relationships. Component Y is deemed to be physically dependent on component X if a malfunction in X can physically damage Y. For example, examination of aircraft accident reports reveals that a disproportionate number of mishaps caused by physical component malfunction involves events such as turbine blades breaking loose and damaging nearby (and sometimes distant) components.

DRAPHYS makes its diagnosis by initially suspecting all components that could conceivably be implicated in the malfunction. Each of these fault hypotheses is then tested by determining whether, for every symptomatic sensor, there is a symptomatic path in the functional or physical dependency nets from the suspect component to the sensor. A symptomatic path is one that passes only through components that are either uninstrumented or have symptomatic sensors. DRAPHYS returns as output the set of suspect components that pass this test; the hoped-for result is that this set will be a singleton. It is worth noting that the set of suspects can be pruned dynamically as new symptoms arrive.

Since the functioning of aircraft and spacecraft systems is well-understood, it is generally straightforward (though tedious) to develop the database describing the functional dependency relations. Physical dependencies, however, are a different matter: the possible interactions among components are numerous and unpredictable. The expedient used in DRAPHYS has been to include the most obvious interactions (typically from the turbines and similar energy-bearing components to nearby components) and hope for the best. This approach is adequate for simple models, but becomes intractable for realistic cases. A more systematic approach was required.

Since we are operating on the assumption that the failed component has been diagnosed by DRAPHYS, we can use this information as starting point for the reasoning process. Beginning at the failed component, subsequent events are generated by means of a qualitative spatial simulation, in order to determine possible physical propagation paths. In the next section we describe the nature of this simulation process.

QUALITATIVE SIMULATION OF PHYSICAL FAULT PROPAGATION

We have found that a wide variety of malfunctions of physical systems can be characterized as *leaks*, i.e. the uncontrolled escape of a substance into the environment. Malfunctions such as burst hydraulic or gas lines are, of course, literally leaks. It has also proved useful, however, to treat short circuits as electrical leaks, fires as gas and thermal leaks, and mechanical malfunctions such as explosive decomposition or breakage as leaks of kinetic or potential energy. Our approach, then, is to use knowledge of the malfunction site and its nature, together with a database describing the 3-dimensional extent and composition of physical structures, to simulate the consequences of the leak in question.

At the present stage of research we have implemented the capability to simulate fluid leaks, and have a partial implementation of kinetic energy leaks. (An example of such an energy leak is provided by the turbine disintegration that caused the recent crash of United 231 by propagating to the hydraulic control lines.) We have found that a limited set of principles and constructs has emerged that has allowed the systematic and expeditious creation of qualitative spatial simulations, as well as their extension to new malfunction categories. These constructs are described in the next section.

The Simulation

The qualitative simulation of fault propagation in 3-space (and time) requires the spatial representation of physical structures. This requirement raises problems that are more typical of graphics applications than classical simulation programs. In particular, two broad categories of spatial representation exist: volumetric and boundary representations [Requicha]. Volumetric representations describe an object by systematically subdividing space and describing the content of each subdivision. Boundary representation techniques describe solids in terms of their enclosing surfaces.

The best-known volumetric representation technique is probably oct-trees [Jackins]; boundary representations are more commonly found in applications such as CAD/CAM systems. The current implementation uses a boundary representation technique, since the computations required to perform the simulation are more efficient in this representation. Alternate representations are, however, still under active consideration.

To describe a physical object such as an aircraft or spacecraft, the user enters sets of (coplanar) points in 3-space into the database; each such point determines the vertex of a planar plate. The present system constrains the point sets to be convex polyhedra; the planes defined by such point sets are thus more accurately described as convex polyhedral plates in 3-space. These plates form the surfaces of the volumes to be represented. Furthermore, the user may specify points and volumes that represent *components*, i.e. entities and subsystems that can fail. Malfunctions occur at/in components, and propagate from component to component, either physically or functionally.

Our simulation system is based on a package of procedures for performing a basic set of geometric computations on the representation of 3-dimensional objects described above. These procedures include algorithms to compute the intersection of two or more planes, the intersection of lines and planes, the gradient (downward direction) at a point in the plane, and similar computations. These procedures, in turn, are based on more fundamental routines that find the equation of a plane, given the defining vertices, that determine whether a point is in a plane (i.e. within the polygon defining the planar plate), and similar auxiliary functions. As indicated above, the function library we have developed, while of moderate size, appears to be powerful enough to support an extensive variety of 3-space simulations. We will describe the simulation of the propagation of faults resulting from fluid leaks in some detail, and end by indicating how additional categories of leaks can be represented.

Simulation of Fluid Leaks

As stated previously, a wide variety of malfunctions can be conceptualized as leaks of some type of substance or entity. It was deemed reasonable to begin our investigation by attempting a qualitative simulation of fluid leaks. While such malfunctions are more likely to cause problems via functional rather than physical propagation, fluid leaks can propagate physically by shorting out accessible electrical components, corrosion, and a wide variety of other types of spoilage. A more important consideration, however, was the expectation (justified, as it happens) that the algorithms developed in the process of implementing a qualitative simulation of fluid leakage would form a basis for simulating other kinds of faults as well. By way of example, propagation from gas leaks can be simulated by running the fluid leak simulation twice, the second time with the direction of gravity reversed.

Recall that DRAPHYS produces as output the identity of the initial failed component. Since malfunctions can occur only in components, and since the physical location and extent of each component is stored in the database, we will assume that the exact location of the leak is known. This is in fact a simplifying assumption for the purposes of this discussion, since in most cases the sort of components that can leak fluid will be pipes, which typically extend for considerable distances. A description of each component can be stored in the database, so that the nature of the leak (fluid type, pressure) can be retrieved. For aircraft the leaking fluid will usually be hydraulic fluid or fuel. We make the additional simplifying assumption that the fluid is not under high pressure (else techniques more appropriate to energy leaks become appropriate), that there are no complications such as phase changes or leakage into slipstreams, and that the leaking fluid remains inside the air/spacecraft (we cannot simulate "blue ice" at this stage of the game).

We thus have a fluid leaking into the vehicle interior from a known point in 3-space. The simulation proceeds by determining the surface to which the leaking fluid will drop, found by dropping a perpendicular from the leak point. The gradient at the point where this perpendicular intersects the topmost surface under the leak point determines the direction the liquid will take. The path of the liquid from this point is determined computationally, until a level is reached from which, in intuitive terms, there is "nowhere to go but up"; more formally, until the minimum point or plane of an upward concavity is encountered. At this stage the algorithm proceeds to simulate the mounting fluid level, creating a new horizontal plane a small increment Δx above the above-mentioned (local) minimum, and determining the intersection of the new plane (called a *level*) with the planes that form the side of the upward concavity. If this intersection contains points that are outside the plates that form the concavity (it suffices to check points in the intersection of the level and the edges of the concavity), then we have found a level at which the fluid will spill out of the concavity. The spill point is determined, treated as a new leak source, and simulated in the same manner as the preceding sources. Otherwise, a new plane Δx above the previous one is created to represent the advancing fluid level, and the process iterates.

When a new level-representing plane is generated, it is determined whether it intersects the space occupied by a component. If so, a possible propagation path from the original leak to the component is recorded. The effects of failure of the component that was reached can then be propagated further along the functional dependency net.

It should be noted that the paths thus determined are *possible* rather than predicted propagations. Whether such a propagation actually occurs depends on numerous factors such as leak flow rate, amount of fluid available, etc. In most cases of malfunction such factors can only be approximated; furthermore, the amount of time required for these possible events to occur, and in some cases even the ordering of the event sequences, is not predicted by the simulation. The predictions made by the simulation are thus inherently *qualitative* in nature.

The above description of the simulation procedure, while conceptually straightforward, glosses over a large number of computational complexities. The physical world is a complicated place, with behavior determined largely by local interactions. Attempting a qualitative simulation of three-dimensional events based on non-local computations entails a large number of special cases, most of which had to be found the hard way. It is surprisingly difficult, for example, for the simulation to establish and keep track of which side of a plane the fluid path belongs. Alternate representations based on local rules, as described in [Gardin] and [Taylor], were considered but tabled on the grounds of computational intractability.

CONCLUSION

We envision the final form of the spatial simulation system described above as part of an interactive pilot aid system that allows the human to remain in the loop, rather than attempting to solve all problems of diagnosis, prognosis, and recovery planning within the program. The motivation for this orientation lies in our belief that attempting to construct a completely autonomous system would confine its scope of operation to "toy problems", an ever-present bane of AI systems. It is expected that the operator will have the capability of posing a wide variety of queries to the system, including queries regarding possible physical or functional fault propagation paths. The system will provide answers based not only on database retrievals, but also on qualitative simulations such as the one described in this paper, as well as an array of alternate qualitative and quantitative reasoning techniques.

The reasoning processes of AI programs are typically based on deductive inference mechanisms. A number of powerful techniques for such inferencing have been developed, but tend to suffer from representational difficulties, particularly the frame problem. As discussed in [Sloman], reasoning based on analog rather than Fregean representations can offer a way around many of these problems.

Simulation-based reasoning of the sort described in the present paper, as well as the work of [Taylor] and [Gardin], represent explorations in reasoning techniques based on analogical representations.

REFERENCES

Abbott, K. H. (1988), Robust Operative Diagnosis as Problem Solving in a Hypothesis Space, *Proceedings of the 7th National AAAI Conference on Artificial Intelligence.*

Gardin, F, et al. (1986), The Analogical Representation of Liquids in Naive Physics, *Proceedings of the 7th European Conference on Artificial Intelligence, (ECAI-86), Brighton, U.K.*

Jackins, C. L., and S. L. Tanimoto (1980), Oct-Trees and Their Use in Representing Three-Dimensional Objects, *Computer Graphics and Image Processing , Vol 14, 249 -270.*

Requicha, A. (1980), Representations of Solids: Theory, Methods, and Systems, *Computing Surveys, Vol. 12, No. 4.*

Schutte, Paul C. (1989), Real-time Fault Monitoring for Aircraft Applications using Quantitative Simulation and Expert Systems, *Proceedings of the AIAA Computers in Aerospace VII Conference.*

Sloman, A. (1986), Why we need Many Knowledge Representation Formalisms, *British Computer Society Expert Systems Conference.*

Taylor, Hadley C. (1986), Studi sulla Modellazione Computazionale nella Fisica Naive, *M.S. Thesis, University of Milan.*

KNOWLEDGE-BASED AND INTEGRATED MONITORING AND DIAGNOSIS IN AUTONOMOUS POWER SYSTEMS.

J. A. Momoh Z. Z. Zhang

Department of Electrical Engineering
Howard University
Washington, D. C 20059.

ABSTRACT

This paper presents a new technique of knowledge-based and integrated monitoring and diagnosis (KBIMD) to deal with abnormalities and incipient or potential failures in autonomous power systems. The KBIMD conception is discussed as a new function of autonomous power system automation. Available diagnostic modelling, system structure, principles and strategies are suggested. In order to verify the feasibility of the KBIMD, a preliminary prototype expert system is designed to simulate the KBIMD function in a main electric network of the autonomous power system.

Keywords: Expert systems, Failure diagnosis, Power systems.

INTRODUCTION

An autonomous electric power subsystem is one of the most important parts in many automatic systems, including space stations. The safety of the power subsystem depends on the working status of electrical components distributed on all hierarchical levels. Various monitoring and diagnostic measures have been used to deal with abnormalities, failures and faults in the subsystem. They include periodic manual testing, automatic monitoring and testing, and protective relaying. Continuous endeavors have been made to improve the measures; however, difficulties continue to be experienced with some faults, particularly incipient or potential failures of the electrical components.

Building a diagnostic expert system embedded in the software package for the automatic systems, a combination of artificial intelligence (AI) methodology and integrated utilization of status information opens up a new possibility to enhance monitoring and diagnostic techniques usable in the power subsystem. This is a knowledge-based and integrated monitoring and diagnosis (KBIMD) function which can serve as a new function of autonomous power subsystem automation.

This paper presents the KBIMD approach including available diagnostic modeling, diagnostic system structure, diagnostic principles and diagnostic strategies. In order to verify the feasibility of the KBIMD, a preliminary expert system prototype is designed to simulate the KBIMD function in the main electrical network of a power system.

KBIMD CONCEPTION.

The KBIMD is designed to provide continuous monitoring and diagnostics of a real-time systems. It provides an early and more complete revelation of malfunction, abnormalities and failures.

The potential benefits of the KBIMD includes efficient capability to utilize information over a wide scope of equipment, and structural, functional and behavioral knowledge of equipment, and systems. Furthermore, the KBIMD combines the experience of human experts with the computer-based approach to develop an innovative approach to the development of new diagnostic principles and methods.

I. KBIMD STRUCTURE

The proposed KBIMD system is shown in Fig. 1. Its mainframe comprises a systems data base (SD), an expert system (ES) and a data coupling processor (DCP). It is connected with data acquisition units, protective units and, if needed, other diagnostic subsystems. The SD contains (a) current messages of diagnosed objects, (b) historical records of the objects, (3) messages from protective relays and other diagnostic systems. The ES is the main body used to perform KBIMD. The DCP is the coupling part of the ES and SD, and performs the mapping of numerical values into qualitative values.

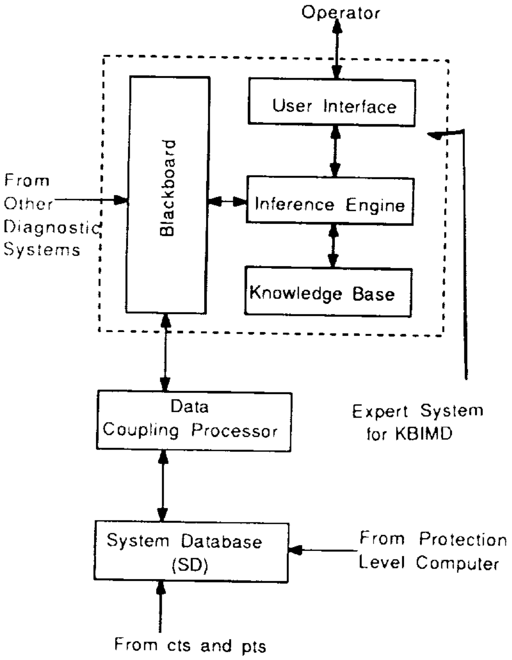


Figure 1. The KBIMD system architecture

II. KBIMD OBJECTS AND MESSAGE SOURCES.

The KBIMD objects consists of components of the electric network autonomous power system. These objects represent the diagnosed quantities of the expert system. It includes current transformers (CTs), and potential transformer (PTs). The CTs and PTs are used for monitoring and protecting relaying subsystems. They provide a source of information to the expert system and by using protective relaying between subsystems. Different fault types and symptoms are measured by the KBIMD systems.

III. PROTECTIVE RELAYING SYSTEMS.

The role of relays in dealing with electrical fault types and symptoms discussed in KBIMD are different even though they are on the same network. However, the message interchange between the two systems are of potential benefit.

IV. HUMAN OPERATORS AND OTHER DIAGNOSTIC SUBSYSTEMS

With the aid of human operators and other diagnostic subsystems, information on causes of failures are identified. This information provides the basis for developing the expert system.

COMBINATORY MODELING FOR KBIMD.

Three knowledge-based models are designed to provide intelligent diagnostic reasoning of faults. They are discussed under physical configuration model PCM, hierarchical studies model (HSM), and the cause-effect relation model (CERM) of the electric power subsystem. The illustrative network shown in figure 2 below is used as an example.

I. PHYSICAL CONFIGURATION MODEL (PCM)

A component in the subsystem can be diagnosed by the ES only when it is live. All live components constitute the live part in the electrical network of the subsystem. The PCM is used to represent the existing physically relational situation of the network, including joint relations, connected status, live status and operational status of all components in the network. It provides the KBIMD with a clear description of the live parts in the network and their changes with operational requirements.

II. HIERARCHICAL STRUCTURE MODEL (HSM).

The HSM is a hierarchical structure description of the subsystem to enable hierarchical failure-search in the KBIMD. It is organized in the following way:

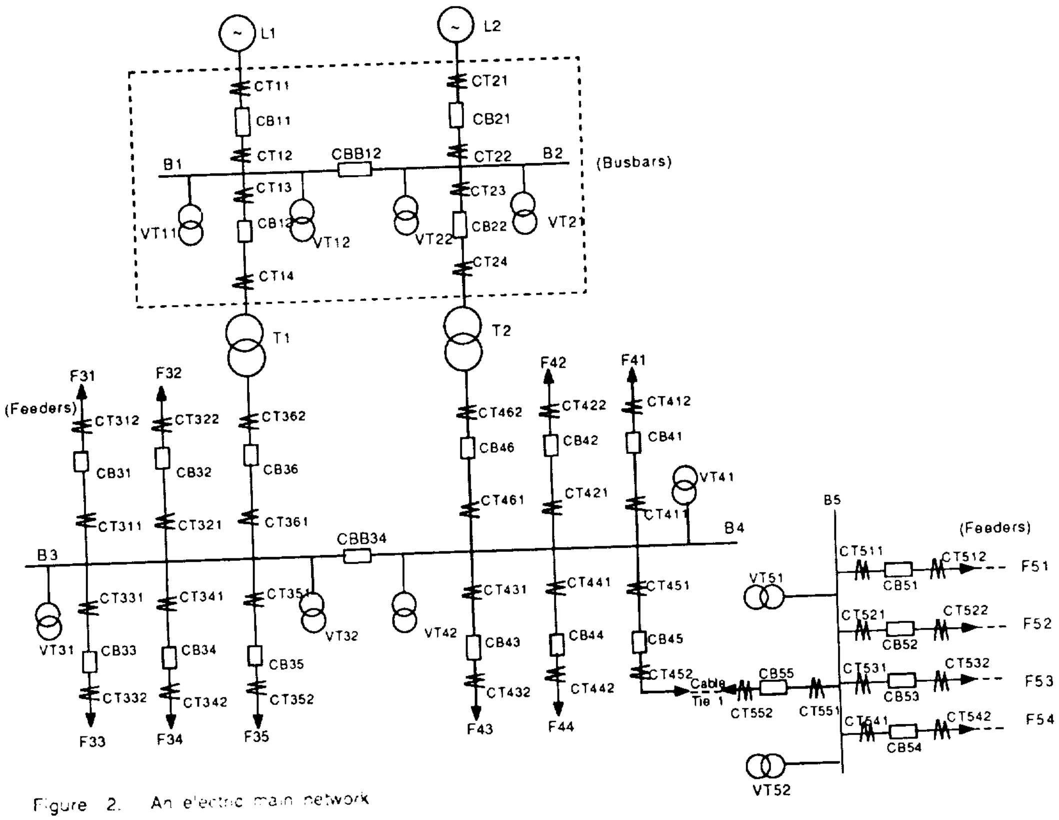


Figure 2. An electric main network

A. The power system is treated as a multi-level and hierarchical diagnosed system,

B. The diagnosed system is divided into separated subsystems at each hierarchical level,

C. A subsystem consists of one or several diagnosed components which have some kinds of characteristic relations,

D. An abnormal or faulted component to be diagnosed is a subsystem at the lowest level.

For example, assuming that the part enclosed with a dotted line in Fig. 2 is live, a description of the HSM is as shown in Figure 3.

III. CAUSE EFFECT RELATION (CERM).

In order to efficiently perform diagnostic reasoning, some experimental and heuristic diagnosis knowledge is integrated into the KB to speed up the failure search. The CERM employs a semantic network approach (Toransso, et al.,1987) coupled with the search for failure by using indirect relationships between failure and symptoms. In addition, CERM bridges the gap between electric symptoms and failures in non- electric parts.

COMBINATORY DIAGNOSIS PRINCIPLES IN KBIMD.

To achieve a speedy and accurate implementation of diagnostic reasoning in the ES suggested for KBIMD, a combinatory diagnosis KB scheme is developed. The functions are discussed in four different subsystems below.

1. DIAGNOSIS BASED ON FIRST PRINCIPLES (DBFP).

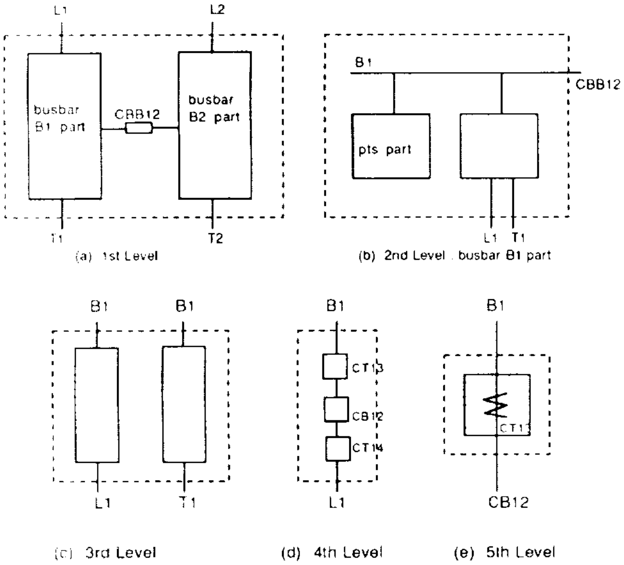
The DBFP subsystem obtains information from structural description of diagnosed objects quantities and behaviors. This subsystem employs a validation check on physical laws to pinpoint the existence of failures.

For example, in Figure 2, the sum of primary currents at CT1 or CT2, CT21 or CT22, CT14 or CT13, CT24 or CT23 are checked on the same phase conductors and checked for zero measurements at normal conditions. If non-zero values are obtained, a failure signal is flagged.

2. DIAGNOSIS BASED ON STRUCTURE (DBSK)

The diagnosis is based on the multilevel and hierarchical structure (HSM) of the electric power subsystem. It begins with the highest level of the HSM and moves to the lower level in the model. As in previous levels, it employs first principle and experimental knowledge as tools for its diagnostic reasoning. This (DBSK) is capable of narrowing down possible failure to a low level within a small region. The application of this structure-based diagnosis scheme is demonstrated for failure of CT13 in Figure 2.

The sequence of diagnostic reasoning in a multilevel sequence is shown in Fig 3. It illustrates the failure search pattern from level I to level V.



(a) 1st Level (b) 2nd Level . busbar B1 part

(c) 3rd Level (d) 4th Level (e) 5th Level

Figure 3. A HSM example discription

III. DIAGNOSIS BASED ON FUNCTION KNOWLEDGE (DBFK).

The function-based diagnosis is employed only when failure search has been narrowed to a suspected component. It is the functional relation and model of diagnosed component. The DBFK identifies suspected failures or eliminates a suspicion. In the latter, this suspicion is recorded as a failure disturbance. The recorded components are available for subsequent diagnosis.

IV. DIAGNOSIS BASED ON EXPERIENTIAL KNOWLEDGE (DBEK).

Experiential knowledge of human experts is based on their diagnostic practice over a lengthen period of time. This allows them to diagnose failure faster, accurately and efficiently. The DBEK employs the following different strategies to construct the knowledge bases.

(a). Identification Based on Comparison.

This involves cross comparison between a given component of the same type with same input. If one of them is faulty, the observation will yield different results. The second is the self-comparison approval which compares the components with current observation on a component with its historical record. The difference is used to verify the possibility of a fault. The third approach removes a component part of the HSM system and checks if it leads to a failure-free system, and then recommendation of the fault situation is suggested.

(b). Determination of Diagnostic Ordering.

When diagnostic reasoning is exhausted, further diagnostic reasoning is needed to execute the experience of failure probability. The diagnostic ordering scheme identifies components guaranteed to fail.

(c). Discrimination Based on Historical Record.

When recent historical records on components manifest repeated "failure disturbance." It is certain that a fault exist in the component.

(d). Discrimination Based on the CERM.

Some failures have their origin in the cause-effect relationship and their discovery is based on diagnostic experience. Discrimination based on the CERM principles may be used to speed the diagnostic reasoning by directly pointing to possible failure sources.

IMPLEMENTATION STRATEGY.

Several basic strategies are used for implementing the KBIMD. The strategies may be modified to suit diverse requirements of an autonomous power system with alternating or direct power source. The strategies are divided into three major parts.

1. Failure Monitoring.

Prior to failure search by the ES, all subsystems at the highest level of KBIMD are monitored in a circular or repeated manner. The monitored electrical quantity must satisfy current and voltage balance relations. For example in Figure 4, the current balance relation using Kirchoff's 1st law gives:

$$\sum_{n=1}^{4} i_n = 0 \qquad (1)$$

where i_1 through i_4 shown in figure 4 denote the input/output port current of a given phase:

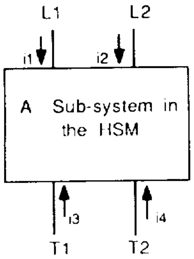


Figure 4. Ports currents of a sub-system

Similarly, for the voltage balance relations, we employ Kirchoff's 2nd law to validate voltage relations at normal conditions that is:

$$V_1 = V_2 = V_3 = V_4 \qquad (2)$$

where V_1 through V_4 are phase voltage on its input/output ports. It should be noted that the currents and voltage quantities are stepped down values obtained from current and potential transformer CTs and PTs.

B. SINUSOIDAL WAVEFORM PATTERN OF CURRENT SUM

Sinusoidal waveform pattern recognition is based on typical characteristics of alternate currents. For example, the subsystem in Fig. 4 should satisfy the following waveform pattern under normal conditions:

$$\sum_{n=1}^{4} |i_n| = |A.\sin(\omega_f t + \theta| \qquad (3)$$

where w_f is the fundamental frequency, A and θ are real constants. Equation (3) means that a sum current of all port currents should have a sinusoidal waveform under normal conditions.

FAILURE SEARCH.

Failure search is performed by the inference engine in the expert system and advances to locate the failure in the power system. In a failure mode, it employs the service of DBFP which employs equations (1), (2), and (3) to determine whether a diagnosed system contains a possible fault source or not.

To narrow the faulty region into as small an area as possible DBSK system on the HSM is used. It narrows down the fault region into the smallest area possible. The DBEK system is used to assist the reasoning to detect possible faulty components more quickly and accurately. While the DBEK is used to verify diagnosis results and to determine the failure types. Qualitative reasoning is performed to implement description of equations (1), (2) and (3).

II. DATA KNOWLEDGE MAPPING.

The diagnostic reasoning in the ES is based on real-time data in the system data base which represents a stepped down version of fault quantities. To implement the diagnostic reasoning, data coupling between symbolic computation for the ES and the numerical computation is required. The implementation procedure is as shown in Fig 1.

The DCP structure performs transition from numerical values into qualitative values. The data knowledge mapping is based on selection of one of the following modes:

(a) Three quantitative ranges of "balanced," "unbalanced," "high unbalanced" condition of three phase voltages and currents.

(b) Four qualitative ranges, "Zero," "low," "high" and some for comparison of voltages and currents.

(c) Three qualitative ranges, "zero," "near zero" and "not zero"from equation (1).

(d) Three qualitative ranges "equal," "unequal" and "highly unequal" for equation (2).

(e) Two qualitative ranges "normal" and "abnormal" in equation (3).

EXPERT SYSTEM ON KBIMD.

A preliminary prototype of the ES used to verify feasibility of the KBIMD in autonomous power systems has been designed in PROLOG[4]. Its structure, shown in Fig. 1, comprises the four ports: knowledge base, blackboard, inference engine and user interface.

KNOWLEDGE BASE

The knowledge base developed for the KBIMD consists of a fact base and a rule base. The fact base contains the fact statements which describes the behavior and records of all components. It stores qualitative knowledge of real-time message sources and solution procedures for handling diagnostic problems.

The status and descriptions of PCM and HSM are given in the fact base. The rule base consists of IF-THEN statements. Using production rules, the basic decision-making gives diagnostic reasoning in the KBIMD. The rule base contains rules for forming and changing the PCMs and HSM. It also presents rules for CFRM, and gives description rules for DBSK, DBFK and DBEK.

II. BLACKBOARD.

The blackboard approach uses data base for message communication between the ES and the outer units. The blackboard provides messages or order or starting or stopping rules for DCP. The DCP gives qualitative value and issues messages from other diagnostic subsystems.

The blackboard consists of blackboard Monitor (BM), Input-Blackboard (IB) and Output-Blackboard (OB). Its structure is shown in Fig. 5. BM is a part of the inference engine which maintains and controls access to the blackboard. OB is used to provide the DCP with necessary messages for selecting relevant data in the SD and performing needed numerical computation to implement the diagnostic strategies. IB is used to receive qualitative values from DCP which are necessary for performing qualitative reasoning in the KBIMD.

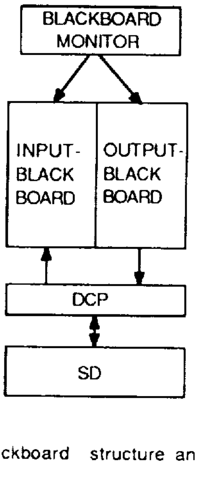


Figure 5 Blackboard structure and interaction

Inference Engine

The inference engine is the part in the ES which contains the general KBIMD problem-solving knowledge. It uses the domain knowledge in the knowledge base and performs message interaction on the blackboard. It consists of hierarchical and modular procedures, and is based on data-driven, forward chain and meta-rules methods.

The inference engine is designed to determine and evaluate the working of the ES forms and changes in the PCM and HSM. It controls and utilizes the blackboard for monitoring and starting failure search to locate possible failure. The record of diagnostic and the approach suggested for handling failure types is also given.

The inference engine and a user interface form an expert system shell constructed in a multi-level and hierarchical form shown in Figure 6.

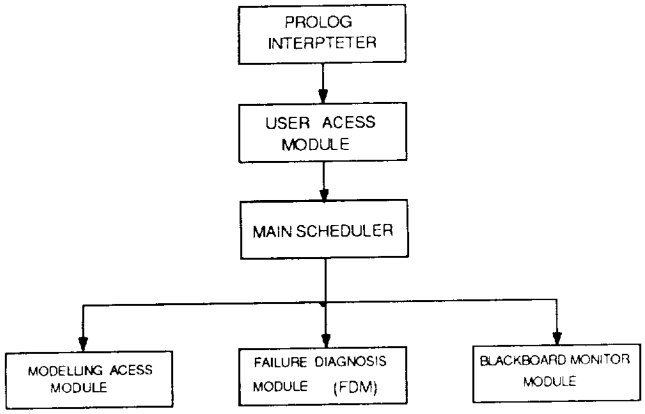


Figure 6. Modular structure of the expert system shell

Failure-Diagnosis Module (FDM) is constructed by several sub-modules, shown in Fig. 7. The KBID (Knowledge-Based and Integrated Diagnosis) scheduler is based on data-driven and meta-rules methods. Data from the main scheduler or other modules are based on reasoning steps. Meta-rules are used for scheduling of sub-modules at the lower level through ordering and utilizing of rules. As soon as a diagnosis has been completed, the reasoning process will return from the Result Output Module to the User Access Module to report a diagnostic result to the operator.

SIMULATION TEST EXAMPLES

Several test examples are presented to illustrate the feasibility of the KBIMD and the application of the ES. The examples are based on the electrical network configuration in Fig. 2. Simulation tests are performed in the following way:

(1) Set the ES in a waiting state,
(2) Input qualitative values necessary for KBIMD into the blackboard,
(3) Set the ES in an operational state,
(4) The ES tells failure locations as its diagnostic results,
(5) The ES explains its diagnosis through man-machine dialogue.

I. Example I

Location : VT21 (when CBB12 closed)

Failure: secondary winding of A phase in turn-turn short circuit.

Input: (1) three-phase voltages of VT21 "unbalance"
(2) the A phase voltage "low," B and C phases of VT21 "same".

Result and Explanation:

A failure inside the A phase of VT21 secondary part.

BECAUSE OF three-phase voltages of VT21 "unbalance;"

AND the A phase voltages of VT11, 12, and 22 "equal", and of VT21 "unequal";

AND B, C phase voltages "same" and A phase of VT21 "low".

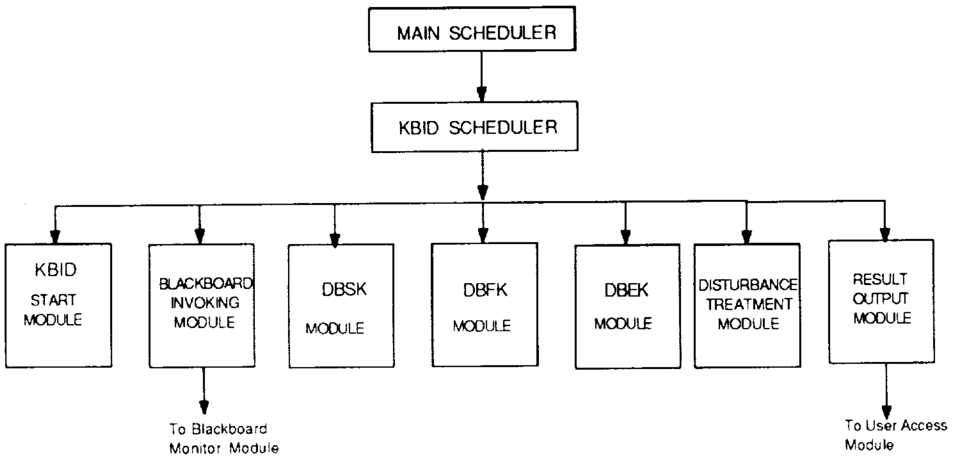


Figure 7. Structure of the FDM

II. Example II

Location: CT321 (when CBB34 opened)

Failure: secondary winding of B phase in open condition

Input: (1) three-phase currents of CT321 "high-unbalance;"
(2) B phase current "zero," A and C phase currents of CT321 "same."

Results and Explanation:

A failure inside the B phase of CT321 secondary part.

BECAUSE OF B phase sum current of CT311, 331, 341, 351, 361, and 321 "not zero;"

AND the sum current of CT311, 331, 341, 351, 361 and 322 "zero;"

AND A, C phase currents "same" and B phase current of CT321 "zero."

III. Example III

Location: Cable tie 1 (when CBB34 opened)

Failure: a partial discharge in C phase-ground of tie 1.

Input: (1) C phase currents "same," waveforms of CT451, CT452 "abnormal;"
(2) C phase current waveform of CT552 "normal."

Result and Explanation:

1. A failure inside the C phase of Cable tie

BECAUSE OF C phase current waveform of CT451 and 452 "abnormal;"

AND C phase current of CT451 and 452 "same;"

AND C phase sum current of CT411,421, 431, 441, 451, and 461 "zero;"

AND C phase sum current of CT411, 421, 431, 441, 461, and 552 " not-zero;"

AND C phase current waveform of CT552 "normal."

CONCLUSION.

The autonomous electric power system is one of the most important parts in many automatic systems. Malfunctions, abnormalities and incipient or potential failures in the autonomous electric power system have been a difficult problem to address. With the application of expert system technology and integrated utilization of information, this paper suggests an approach of knowledge-based and integrated monitoring and diagnosis (KBIMD) to deal with the failures in autonomous power systems. The paper presents:

(1) The KBIMD basic conception and available system structure scheme of the KBIMD,

(2) Combinatory modeling for the KBIMD which is performed through a combination of a physical configuration model, a hierarchical structure model and a cause-effect relation model.

(3) Combinatory diagnosis principle for the KBIMD which is performed through a combination of diagnosis based on first principles, structure knowledge, function knowledge and experiential knowledge,

(4) Basic implementation strategies for the KBIMD,

(5) A preliminary design of the prototype expert system used for the KBIMD.

The paper gives simulation test examples to illustrate the feasibility of the KBIMD and the prototype expert systems.

Further research will be necessary to advance the KBIMD suggested here to practical application in autonomous power systems. It should include:

a Development of information integration utilization methodology

b Development of diagnostic principles available to no-electric parts in autonomous power systems

c Knowledge-based recovery after completion of a diagnosis process

d Full design and implementation of a practical KBIMD system.

ACKNOWLEDGEMENTS

This work was partly supported by the United States Department of Energy, the National Science Foundation under grant ECS 8657559PYI Award, and City of Los Angeles Department of Water and Power. The authors also wish to thank the staff of the Energy Systems Network Laboratory, especially Mr. Anthony Akande for his assistance in the preparation of the manuscript.

REFERENCES

P. Torasso, L. Console, (1987). "Causal Reasoning in Diagnostic Expert Systems", *Proc. of SPIE, vol. 786 applications of Artificial Intelligence* V , pp 598-605.

R. Reiter, (April 1987). "A Theory of Diagnosis from First Principles", *Artificial Intelligence*, 32 , pp. 57-95.

R. Milne, "Strategies for Diagnosis", (May/June 1987). *IEEE Trans. on System, Man and Cybernetics*, vol. SMC - 17, n.3, , pp. 333-339.

W. F. Clocksin, C. S. Mellish, (1984). "Programming in PROLOG", *Springer-Verlag*, Berlin.

THE PROCEDURAL SAFETY SYSTEM

Maureen E. O'Brien

Goddard Space Flight Center
Greenbelt, Maryland

ABSTRACT

Telerobotic operations, whether under autonomous or teleoperated control, require a much more sophisticated safety system than that needed for most industrial applications. Industrial robots generally perform very repetitive tasks in a controlled, static environment. The safety system in that case can be as simple as shutting down the robot if a human enters the work area, or even simply building a cage around the work space. Telerobotic operations, however, will take place in a dynamic, sometimes unpredictable environment, and will involve complicated and perhaps unrehearsed manipulations. This creates a much greater potential for damage to the robot or objects in its vicinity. The Procedural Safety System (PSS), developed at GSFC's Robotics Laboratory, collects data from external sensors and the robot, then processes it through an expert system shell to determine whether an unsafe condition or potential unsafe condition exists. Unsafe conditions could include exceeding velocity, acceleration, torque, or joint limits, imminent collision, exceeding temperature limits, and robot or sensor component failure. If a threat to safety exists, the operator is warned. If the threat is serious enough, the robot is halted. The PSS, therefore, uses expert system technology to enhance safety thus reducing operator work load, allowing him/her to focus on performing the task at hand without the distraction of worrying about violating safety criteria.

Introduction

As we move from industrial automated robot applications toward telerobotic operations, particularly for space applications, the need for a sophisticated safety system increases dramatically. Industrial automated robots, which traditionally involve repeating pre-programmed "pick and place" operations, utilize unsophisticated sensing capabilities and typically incorporate a very limited amount of safety since each point that the robot is supposed to move to is pre-programmed. Telerobotics which involve both autonomous and teleoperated control performing a wide variety of tasks utilizing many different sensing capabilities must incorporate a great deal of safety because the motions of the robot are, for the most part, variable. A robot in a manufacturing plant, for example, may be tasked to drill a 1/2 inch hole in a sheet of metal. Every point that the robot is supposed to go to in order to drill the hole has been predetermined. Safety checks that are sometimes used involve using a sensor to detect if a human has entered the work area of the robot or using a sensor to detect if a robot has stopped its motion.

The Flight Telerobotic Servicer (FTS), the robot which will be used to service the Space Station Freedom, will be tasked to do a wide variety of tasks such as refueling a satellite, repairing a satellite and assembling the trusses for the Space Station. These types of tasks, unlike traditional industrial automated robot tasks, incorporate both autonomous and teleoperated control utilizing a great deal of sensing capabilities, requiring sophisticated safety systems. There are several functions that a complete safety system for telerobotic operations must incorporate. First, the safety system must be able to detect unsafe robot commands being sent from the robot control computer to the robot. Second, the safety system must be able to detect unsafe robot health status to ensure the robot is not malfunctioning. Third, the safety system must monitor all other systems such as the workstation computer, sensors, and robot controllers to ensure that they are operating. Finally, the safety system must be able to monitor all sensor data to ensure the task is operating under safe conditions. All of these functions must be incorporated to ensure the safety of humans, the robot and the objects in the robot environment.

Overview of the Safety Problem

These functions can be divided into two safety systems: the Watchdog Safety System (WSS) and the Procedural Safety System (PSS). The WSS provides safety at the robot servo level. The WSS is a separate system which exists between the robot control computer and the robot. It monitors all commands sent from the controller to the robot to ensure that the following have not been exceeded:

- velocity limits
- acceleration or motor torque limits
- joint limits.

The Watchdog Safety System, unlike the Procedural Safety System, checks absolute limits. It, for example, checks to ensure that the robot never exceeds a velocity limit of 250 mm/sec. The WSS must also monitor all robot status data to ensure that the following are not present:

- temperature limits exceeded
- incorrect position reached.

All other systems such as the sensors, the robot and the workstation computers must also be monitored by the WSS to ensure that they are operating.

This paper focuses on the Procedural Safety System (PSS) developed at the Goddard Space Flight

Center's Robotics Laboratory which is an expert system which provides safety for operating the FTS. The PSS exists at a higher level than the Watchdog Safety System. It, unlike the WSS which checks absolute limits, detects unsafe conditions based on the operational limits of the step of the task. Sensor data and commands are sent to PSS which checks this data against the operational limits of the task as shown in Figure 1. The PSS, which exists between the operator interface and the robot controller, obtains all sensor data and commands from the sensors and the operator. It compares the data and commands with the operational limits of the present step of the task. If the data or commands lie outside of the operational limits, the PSS sends messages to the operator interface to warn the operator or sends commands to the robot controller to stop the robot motion.

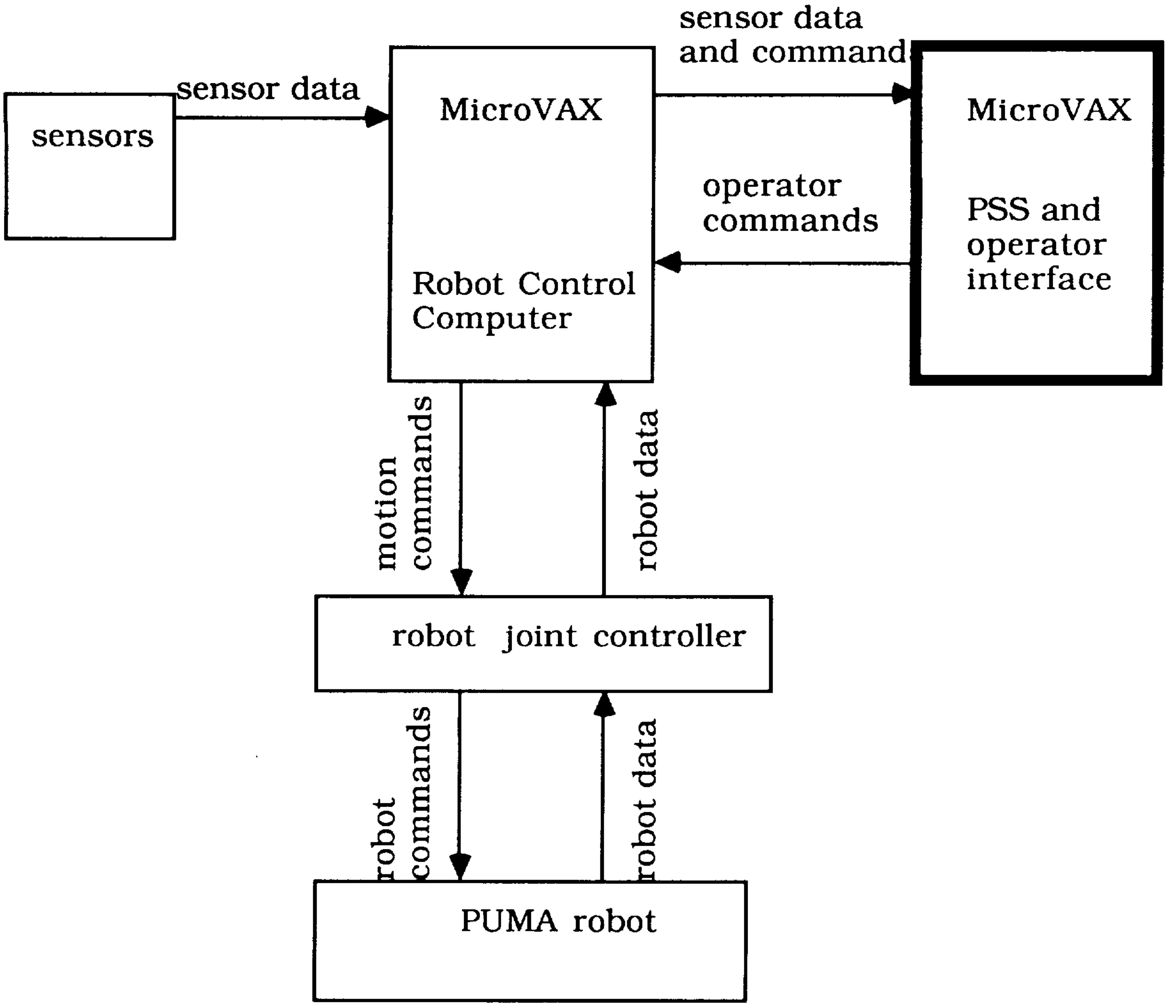


Figure 1. GSFC Robotics Laboratory Procedural
Safety System Layout

The Application

The Procedural Safety System was added to the Orbital Replacement Unit (ORU) demonstration that exists in the robotics laboratory. The ORU demonstration involves using a PUMA robot to move an ORU, a generic housing for a flight experiment, from one position on a platform to a second position. The robot then opens the door of the ORU, allowing the robot to replace submodules or repair the experiment. Prior to the implementation of the PSS, the ORU demonstration incorporated very few safety checks. The burden of ensuring that the task was operating safely was placed on the operator. The PSS relieves this burden by monitoring sensor data and commands and reporting anomalies to the operator.

The ORU task was broken down into steps as shown in figure 2. Each step must be completed and conditions must exist in order to continue to the next step. For example, the gripper must be latched to the ORU handle before the robot can move it to the second position on the platform. Associated with each step of the task are the operational limits for the various sensor data. Figure 3 shows the operational limits for steps one and two. When step one of the task is being executed no forces or torques should exist, the gripper should be unlatched, and the switches on the platform indicting which position the ORU is in should indicate that the ORU is in position one (switches 1,2 and 3 connected and switches 4,5 and 6 disconnected). Operational limits for step two are also shown in figure 4. These operational limits were determined by using a computer program to obtain the minimum and maximum values of the sensor data as the ORU demonstration was performed by several different telerobot operators, multiple times. A certain percentage was then added to these values to account of noise in the readings.

TASK : replace an orbital replacement unit (ORU)

Step 1 : goto point above oru door handle

Step 2: seat on door handle

Step 3: latch gripper to handle

Step 4: move ORU to position 2

Step 5: seat ORU on platform

Step 6: unlatch gripper from handle

Step 7: goto park position

Figure 2. ORU Task Steps

Step 1: goto point above oru door handle

force in x < 0 lbs
force in y < 0 lbs
force in z < 0 lbs
torque about x < 0 in-lbs
torque about y < 0 in-lbs
torque about z < 0 in-lbs
gripper unlatched
platform switch 1,2,3 connected
platform switch 4,5,6 disconnected

Step 2: seat gripper on door handle

force in -5 < x < 5 lbs
force in -5 < y < 5 lbs
force in - 15 < z < 15 lbs
torque about -1 < x < 1 in-lbs
torque about -1 < y < 1 in-lbs
torque about -10 < z < 10 in-lbs
gripper latched
platform switch 1,2,3 connected
platform switch 4,5,6 disconnected

Figure 3. Operational Limits for ORU Replacement Task

Nexpert's Representation of the operational limits

```
Rule:  Rule 33
If
        step.number is precisely equal to 1.00
        And there is evidence of assign_limit_values
Then limit_values_assigned
        is confirmed.
        And task_gripper.status is set to unlatched
        And -5 is assigned to task_lower_force_limits.x
        And 5 is assigned to task_lower_force_limits.y
        And 10 is assigned to task_lower_force_limits.z
        And -70 is assigned to task_lower_torque_limits.x
        And -10 is assigned to task_lower_torque_limits.y
        And -10 is assigned to task_lower_torque_limits.z
        And 20 is assigned to task_upper_force_limits.x
        And 30 is assigned to task_upper_force_limits.y
        And 40 is assigned to task_upper_force_limits.z
        And -10 is assigned to task_upper_torque_limits.x
        And 50 is assigned to task_upper_torque_limits.y
        And 50 is assigned to task_upper_torque_limits.z
        And task_oru_position.position1_status is set to connected
        And task_oru_position.position2_status is set to disconnected
        And assign_limit_values is set to FALSE
        And 17 is assigned to message_num.number
        And Execute dectalk_male(@ATOMID=message_num.number;)
```

Nexpert's Representation of decision process

```
Rule : Rule 16
If
        there is no evidence of force_in_z_approaching_upper_limit
        And task_upper_force_limits.z_forces.z is less than 0.0
Then indicate_unsafe
        is confirmed.
        And force_in_z_approaching_upper_limits is set to TRUE
        And 5 is assigned to message_num.number
```

Figure 4. Nexpert Representations

There are three safety issues that need to be addressed pertaining to telerobotic operations. How should an unsafe condition be detected? After it is detected, what action should be taken to respond to this unsafe condition? What should be done to recover safely from this unsafe condition? In the PSS implementation of the ORU demonstration, we chose to use the expert system shell, NEXPERT, to detect an unsafe condition based on the operational limits. NEXPERT is an object oriented, rule based expert system shell which allows one to represent knowledge in a rule format and reason about this knowledge to solve a problem. We chose to use NEXPERT for three reasons. First, after evaluating several other expert shells such as Clips and KEE, NEXPERT was

the best expert system shell for the money. Second, the safety problem lent itself to the rule format, for example, if force in x is greater that 10 pounds then notify operator of unsafe condition. Finally, NEXPERT is object oriented which means that it performs operations on objects depending upon the state of the world. Each control cycle, NEXPERT evaluates only those rules which contain objects which pertain to the state of the world. For example, if the object force_x reaches its operational limits then the rules which contain that object will be evaluated. This differs from procedural languages such as C and Pascal because procedural languages perform operations sequentially as they exist in a procedural language program. NEXPERT receives sensor data and commands and compares these data and commands to its knowledge of the operational limits of the step of the task to determine if an unsafe condition is present.
NEXPERT'S user interface was used the to load the rules into NEXPERT's knowledge base. Figure 4 is an example of the output from NEXPERT's knowledge base after the rules were entered. Rule 33 provides an example of a rule which assigns the operational limits of a step of the task. If the step which involves grabbing the ORU handle is being executed then the hypothesis assign_limit_values becomes true and the operational limits for that step one assigned. Rule 16 provides an example of how NEXPERT determines if sensor data or operator interface commands lie outside of the operational limits of the step of the task. If the force is z is greater than the operational limit for that step of the task then a message number is assigned which will be reported to the operator.

The next issue that needs to be addressed in the area of procedural safety for telerobotic operations is once an unsafe condition is detected, what action should be taken. Currently, in the ORU demonstration, if the PSS detects an unsafe condition it notifies the operator both visually and audibly. Messages are printed to the terminal in the workstation to indicate to the operator which unsafe condition is present. Messages are also sent to the voice synthesizer, Dectalk, which conveys the unsafe message audibly to the operator. Thus, we provide two way of communicating to the operator that an unsafe condition is present. This is necessary since the operator may, for example, be watching the camera monitors instead of the workstation terminal. If and unsafe condition arises and the only form of communication to the user is messages to the workstation terminal, the operator is not going to receive the warning. If the unsafe condition is serious enough that the task should not proceed then the robot motion should stop and the operator should be notified.

Once the unsafe condition is detected and an action is taken, how should the Procedural Safety System recover from the unsafe condition? The PSS that was implemented in the ORU demonstration recovers from an unsafe condition by returning control to the operator to correct the problem. The operator then continues the task at the current step while the PSS continues to monitor the sensor data and commands. Figure 5 summarizes the functions of the PSS.

The Results

The Procedural Safety System that has been implemented in the robotics laboratory has enabled us to begin to look at safety for telerobotic operations. The PSS has made the ORU demonstration easier and much safer to operate. Prior to the implementation of the PSS, the burden of monitoring the sensor data and operator commands was placed on the operator. The PSS relieves much of this burden by monitoring the sensor data and commands from the operator to ensure that the task is operating safely, enabling the operator to concentrate on performing the task itself.

Future Work

There is a great deal of safety related work that needs to be researched and implemented in the robotics laboratory. The effectiveness of the PSS that exists in our laboratory depends upon the amount of sensing capability. At the present time, the robot system is limited by the amount of

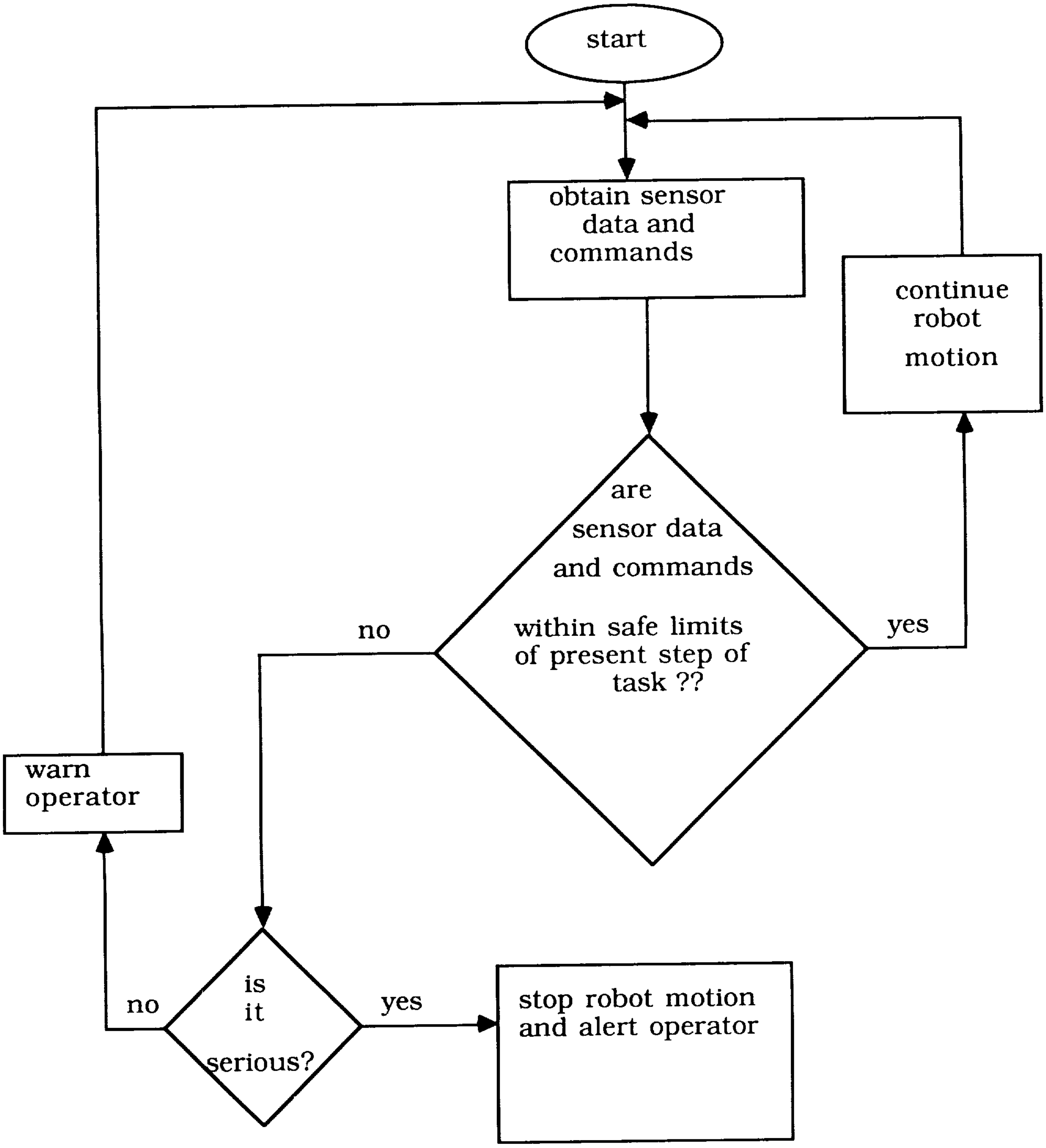

start
obtain sensor data and commands
are sensor data and commands within safe limits of present step of task ??
yes
continue robot motion
no
is it serious?
no
warn operator
yes
stop robot motion and alert operator

Figure 5. Procedural Safety System's Flowchart

sensing capabilities since its only sources of sensor data come from a force torque sensor, the gripper, microswitches and encoder values of the robot. Several other sensing capabilities need to be incorporated. Laser ranging provides 3-D information about an object without having to touch the object. This sensing technique is necessary for object recognition. Tactile sensing provides information about an object between the gripper fingers which cannot be obtained using a force torque sensor because a force torque sensor only provides sensor information at the wrist of the robot. Proximity sensing is another sensing capability that needs to be added to the robot system. It provides quick sensing data indicating the presence of an object which can used for object avoidance algorithms. Besides incorporating addition sensing capabilities into the lab, world modeling techniques need to be researched and implemented which provide a way to organize and represent sensor data so that the PSS can quickly and efficiently acquire the data.

Acknowledgements

The author would like to acknowledge Ms. Susan Cofer who conceived the Procedural Safety System and was instrumental in its development.

Robotics/Intelligent Control

THE JPL/KSC TELEROBOTIC INSPECTION DEMONSTRATION

David Mittman*, Bruce Bon, Carol Collins, Gerry Fleischer,
Todd Litwin, Jack Morrison, Jacquie O'Meara, Stephen Peters
Jet Propulsion Laboratory
California Institute of Technology, Pasadena, California

John Brogdon, Bob Humeniuk, Alex Ladd
Jose Lago, Mike Sklar, James Spencer, Dan Wegerif
Kennedy Space Center, Florida

ABSTRACT

An ASEA IRB90 robotic manipulator with attached inspection cameras was moved through a Space Shuttle Payload Assist Module (PAM) Cradle under computer control. The Operator and Operator Control Station, including graphics simulation, gross-motion spatial planning, and machine vision processing, were located at the Jet Propulsion Laboratory (JPL) in California. The Safety and Support personnel, PAM Cradle, IRB90, and image acquisition system, were stationed at the Kennedy Space Center (KSC) in Florida. Images captured at KSC were used both for processing by a machine vision system at JPL, and for inspection by the JPL Operator. The system found collision-free paths through the PAM Cradle, demonstrated accurate knowledge of the location of both objects of interest and obstacles, and operated with a communication delay of two seconds. Safe operation of the IRB90 near Shuttle flight hardware was obtained both through the use of a gross-motion spatial planner developed at JPL using artificial intelligence techniques, and infra-red beams and pressure sensitive strips mounted to the critical surfaces of the flight hardware at KSC. The Demonstration showed that telerobotics is effective for real tasks, safe for personnel and hardware, and highly productive and reliable for Shuttle payload operations and Space Station external operations.

BACKGROUND*

Telerobotic systems are typically demonstrated with the operator in close proximity to the robot and with nearly instantaneous feedback to direct subsequent actions. However, many applications require ground-based control of remote space-based robots or local control over low-data-rate networks, each of which introduces a significant communication time delay that alters the nature of the operator interaction. Proposed solutions to the time delay problem, including remote site autonomy and a high-level operator interface, need to be tested in an environment where the delays are present.

Inspection tasks are typical of those that will be required of remote robots. One application of telerobotics is for Space Shuttle payload processing. To inspect Shuttle payloads, technicians walk

* The editor can be reached at the Jet Propulsion Laboratory, 4800 Oak Grove Drive, Pasadena, California 91109, Mail Stop 301-250D or at DMittman@Beowulf.JPL.NASA.GOV.

* Adapted from Jet Propulsion Laboratory [JPL], 1989.

above flight hardware to obtain access, and must rely on safety harnesses and expensive temporary scaffolding. A telerobotic system could significantly reduce the cost of payload inspection, and greatly improve the safety of the personnel, the payload, and the Shuttle.

While communication time delay can be avoided in an operational Shuttle "payload inspection robot," autonomous operation and high-level control would improve the cost-effectiveness and safety of the system. By building and demonstrating a prototype that can be controlled either locally or from a remote site, progress in both Shuttle operations and space telerobotics can be achieved.

SPACE FLIGHT PROBLEM DOMAIN

Space Shuttle Payload Operations*

At KSC, access to payloads during pre-launch payload operations is very restricted. At the Operation and Checkout Building, where horizontal payloads are integrated into the payload bay, work-stands are sometimes built to lower technicians down between satellites to retrieve, replace or connect an object. After the integration of the horizontal payloads, the Shuttle is mated to its solid rocket boosters and external tank, and rolled out to the launch pad. Payloads which have to be integrated into the payload bay in a vertical configuration are first inserted into a canister at the Vertical Processing Facility, and are then shipped to the launch pad for integration. When the canister arrives at the launch pad, it is lifted into the Payload Changeout Room (PCR). The PCR is a clean-room integrated into the Rotating Service Structure (RSS); the RSS is rotated against the Shuttle during pre-launch servicing activities.

The payload is first removed from the cannister and brought inside the PCR by the Payload Ground Handling Mechanism (PGHM). The PGHM is a very large device on an overhead beam that removes the payload from the cannister and inserts it into the payload bay. The RSS is then rotated into place in front of the payload bay, and the payload is moved into place.

When the payload has been inserted into the payload bay, it is not visible beyond the PGHM. Limited access to the payload is possible by crawling out onto platforms. "C" clamps, gangplanks and roll-out platforms are used to gain access inside the payload bay. It is sometimes necessary for a technician to climb out onto a gangplank in order to take close-up photographs or to remove lens dust-covers. A technician also has to remove tagged items just prior to launch. Twice for each launch, at the start of PCR operations and at their conclusion, technicians have to reach hazardous positions 65 feet above multi-million dollar payloads to attach grounding straps. This involves bolts and test gear which, if dropped, may cause extensive and costly damage to a payload, requiring removal and repair of the payload, with large "return from pad" consequences.

SPACE FLIGHT OBJECTIVES

One of the objectives of the JPL/KSC Inspection Demonstration was to aid Space Flight operations by demonstrating effective man/machine teamwork on a task that has applications to operational Shuttle

* For a thorough discussion of Space Shuttle Payload Operations see Kennedy Space Center [KSC], 1978.

payload processing. To this end, it was necessary to demonstrate that a telerobotic system can: (a) operate in the complex environment of a Shuttle payload bay or PCR; (b) operate without significant risk to personnel, equipment or payload, reducing both the need for risky gangplank operations, and the chance of errors; (c) improve the productivity of payload operations, easing access to hard-to-reach areas; and (d) improve the reliability of payload operations.

SPACE STATION PROBLEM DOMAINS

Space Station operations for construction and maintenance require extensive access to the external portions of the Space Station. A variety of technologies exist which meet the need for external access, including Extravehicular activity (EVA), Flight Telerobotic Servicer (FTS) teleoperation, and ground-remote telerobotics, each with some advantages and disadvantages.

Extravehicular Activity

The use of EVA involves astronauts in space-suits performing assembly and servicing tasks outside of the Space Station.

Advantages.

One advantage of EVA for on-orbit construction and maintenance of the Space Station is that the astronauts at the work-site can better perceive problems and their solutions.

Disadvantages.

1. There are many risks to the astronaut performing EVA, including the possibility of death during Space Station construction and operations.

2. Astronaut productivity is lower due to the difficulty of performing dexterous operations in a bulky space suit which limits touch and vision.

3. There are large amounts of expensive astronaut on-orbit time required for EVA tasks, e.g. the required three hour pre-breathing period before exiting the vehicle.

4. Limited dexterity increases the possibility of mistakes and reduces reliability and safety.

FTS Teleoperation

The FTS allows astronauts inside the Space Station to perform teleoperation since teleoperation from Earth is not practical due to the communication delay.

Advantages.

The teleoperation of the FTS within the shirt-sleeve environment of the Space Station eliminates the risk to the astronaut due to EVA.

Disadvantages.

1. The limitations of teleoperation contribute to low astronaut productivity, although there is a significant potential for improvement though telepresence.

2. The teleoperation of the FTS, like EVA, requires large amounts of expensive astronaut on-orbit time.

3. The limited dexterity available with teleoperation, and the potential mistakes, reduce reliability and safety.

Ground/Remote Telerobotics

The use of ground/remote telerobotics allows operators at a ground-based control station to operate semi-autonomous telerobot(s) at the Space Station.

Advantages.

1. Eliminates the risk due to EVA.

2. Enhances productivity by allowing telerobotic operations to proceed continuously as long as there is work to be done, with no work stoppages for crew sleep or delays for pre-breathing. Partial autonomy allows one operator to control two or more telerobots.

3. Minimizes astronaut on-orbit time required for external servicing tasks; all robotic control is performed by ground technicians.

4. Enhances reliability since a telerobot can do repetitive assembly tasks automatically without boredom or distraction.

Disadvantages.

The operator's remoteness from the work-site limits the ability to perceive the work-site, thus making problem-solving more difficult and forcing increased reliance on machine autonomy.

Actual Space Station operations will, most likely, include some mixture of EVA, FTS teleoperation, and ground/remote telerobotics, depending on requirements and available capabilities.

SPACE STATION OBJECTIVES

The task of the JPL/KSC Telerobotic Demonstration was to aid Space Station operations by demonstrating effective remote task execution with a limited band-width, uncertain time delay between the operator control station and the work-site, thus overcoming the teleoperation time-delay problem. To this end, it was necessary to demonstrate that a telerobotic system can: (a) operate when the sensor and actuator systems are remote from the operator control station, when the communication band-width is limited, and when there is a variable communication delay of several seconds; (b) operate in a realistically complex flight hardware environment; (c) operate without significant risk to personnel, equipment, or payload, reducing both the need for EVA, and the probability of errors; (d) improve the productivity of operations in space by reducing the need for EVA thus freeing valuable astronaut time for other activities, by operating from the ground thus utilizing far less expensive ground-based personnel, and by allowing more time (even continuous) on-station; and (e) improve the reliability of space operations by reducing mistakes which might be made during EVA due to boredom and fatigue.

THE JPL/KSC TELEROBOTIC INSPECTION SYSTEM

The Robotics Applications Development Laboratory (RADL) at KSC includes a large ASEA IRB90 robotic manipulator on a track and various support computers for controlling the IRB90 and processing video data for machine vision applications. The ASEA IRB90 is an industrial materials-handling robot with a payload capacity of approximately 200 pounds, and a height of approximately nine feet. The IRB90 has been outfitted with a dual-camera platform. The work-site includes an inert PAM and support cradle in a ground support equipment (GSE) frame. The PAM, Cradle, and GSE frame were all obtained from the manufacturer; the Cradle had flown on a previous Shuttle mission, and was to be maintained in a flight-ready condition.

The Task Planning and Reasoning (TPR) and Sensing and Perception (S&P) subsystems were located at JPL, while the Arm Device Control (ADC) and Video Device Control (VDC) subsystems were at KSC (see Figure 1). TPR (Peters, Collins, Mittman, O'Meara, and Rokey, in press) was implemented in LISP on a Symbolics LISP machine, and used a VAX 11/750 as a network communications gateway. S&P (Gennery, Litwin, Wilcox, and Bon, 1987) was implemented in Pascal on a VAX 11/750 with 240 by 320 pixel frame buffers. ADC and VDC were implemented on a MicroVAX, with serial communications to the VME-based processors which contained the direct hardware interfaces to the IRB90 and the video cameras.

Communication between subsystems took place over DECnet using an application layer called the Network Interface Package (NIP). The work-site, with IRB90, controller, video cameras and frame buffers, was located at KSC in Florida. The operator site, with computer and software providing a graphics operator interface, gross-motion spatial planning and machine vision, was located at JPL in California. Communication between the two sites was over a 9600 baud serial link on a shared network (PSCN), resulting in variable and unpredictable communication delays which average two seconds per round-trip transaction.

The intelligent technology used in the JPL software was primarily transferred from JPL's Telerobot Testbed project. This includes the Network Interface Package (NIP) used for all inter-subsystem communications, the graphical user interface (Mittman, 1988) and gross-motion spatial planner (Collins & Rokey, 1988) used by the TPR subsystem, and the machine vision system used by the S&P subsystem. All software except the NIP required modifications and new interfaces for this task.

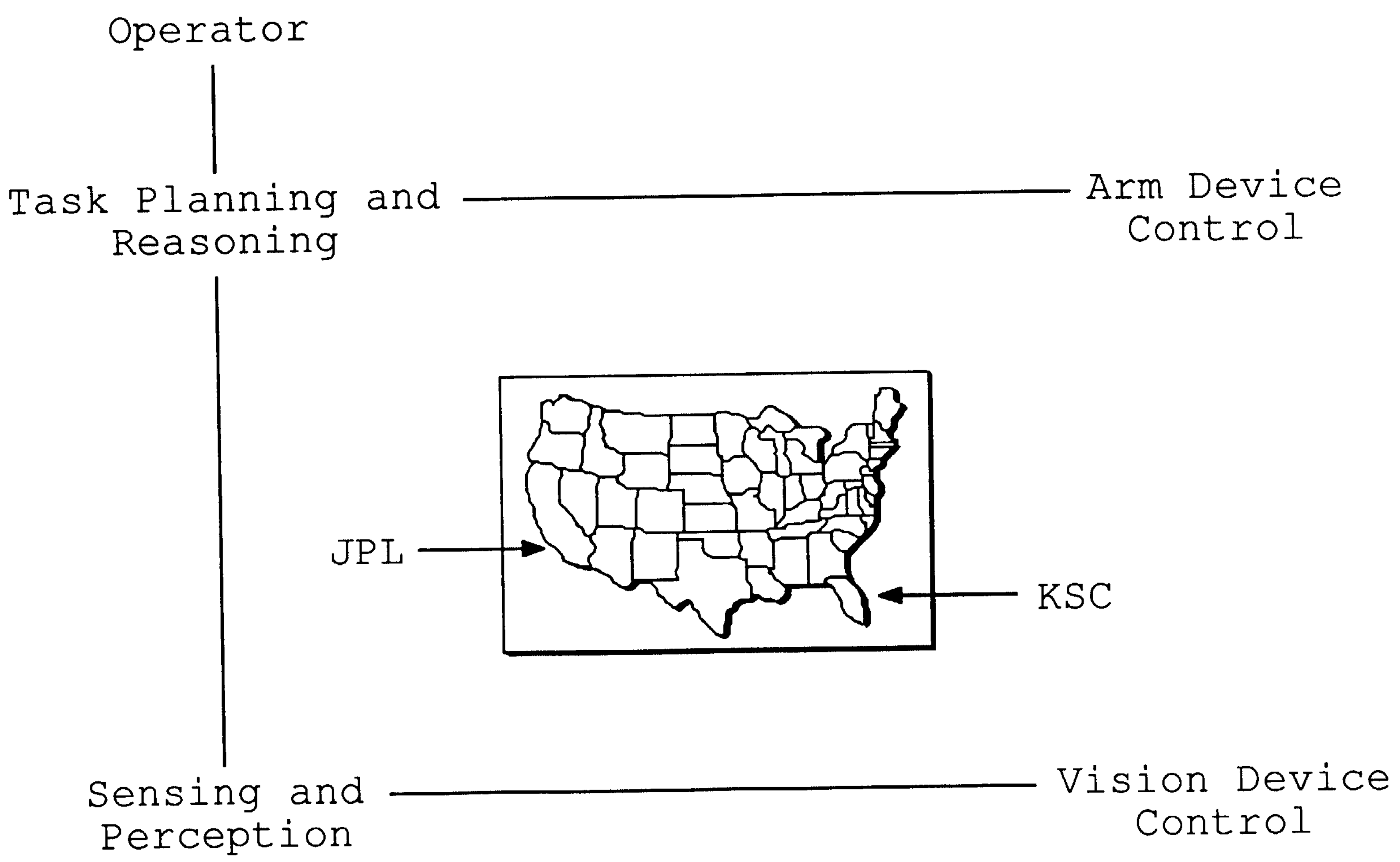
Work-space models for spatial planning, machine vision, and the user interface were derived from a CAD database supplied by KSC. Off-line software utilities at JPL provided transforms to move all models into the same coordinate system and allow calibration of the cameras which supply the images for machine vision. The control station (TPR) commanded S&P to perform its vision functions and also commanded the ADC to carry out the desired robot motions. The S&P subsystem at JPL commanded stereo images to be transmitted from the VDC subsystem at KSC. Using KSC-supplied descriptions of camera viewpoint locations, the S&P subsystem verified the spatial object database required by the high-level spatial planner, thus ensuring the safety of IRB90 motions.

New work performed for this task included implementation of the ADC and VDC subsystems at KSC, generation of IGES models for objects in the work-space, measurement of work-space points to enable calibration, transformation of IGES model data into the IRB90 coordinate frame, conversion of IGES models into the forms needed by the JPL software, generation of free-space maps for use in gross-motion spatial planning, calibration of video camera models for use with machine vision, and implementation of video processing software, including image sub-sampling, compression/decompression, and low-level feature extraction.

SUMMARY OF FIRST-YEAR RESULTS

Hardware and communications were installed, integrated, and tested. The PAM Cradle and inert PAM were acquired and IGES models were

The JPL/KSC Telerobotic Inspection Demonstration



Operator

Task Planning and Reasoning

Arm Device Control

JPL

KSC

Sensing and Perception

Vision Device Control

Figure 1: Logical Block Diagram

created. Device control software was developed at KSC, including communications with JPL software. JPL Sensing & Perception (S&P) subsystem was modified for the needs of this task. The JPL Task Planning and Reasoning (TPR) subsystem was modified and extended for this task, providing gross-motion spatial planning, a direct interface to KSC for IRB90 control and a graphical user interface. A successful capability test was performed, including: (a) control of a robotic manipulator from a distance of 3000 miles with variable time delays averaging two seconds, (b) motion into an occluded, covered region in a very constricted work-space, (c) use of a gross-motion spatial planner to avoid collisions, (d) use of machine vision to verify location of modeled objects, and (e) operation on real flight hardware.

DIFFICULTIES OVERCOME

As might be expected in the first year of a task, numerous difficulties and delays arose. PSCN mistakenly installed a synchronous line instead of an asynchronous line, and the Symbolics NIP version proved to be unusable due to compiler incompatibilities with a new operating system. A VAX NIP server with a custom interface between the VAX and Symbolics machines was created.

The IRB90 controller was of limited use because the proprietary nature of the information contained within the controller made it impossible to obtain accurate IRB90 kinematic parameters. An IRB90 kinematic model was constructed from the IRB90 printed documentation.

The IRB90 controller interface did not accommodate joint controlled motion, the mode used by JPL's gross-motion spatial planner. Motions were planned in joint space, then passed through the forward kinematics of the IRB90 model to derive Cartesian end-effector positions for commanding.

Software was implemented to convert IGES model data and transform it into the IRB90 coordinate frame. Limitations in the CAD system from which the IGES data originated required that conversion software be written with operator interaction to aid in designating IGES object connectivity. Additional software was implemented to compute a homogeneous transformation between IGES model and IRB90 coordinate systems when given a set of points measured in both frames.

Camera calibration within the S&P subsystem was conducted with a poor dispersion of calibration points. The iterative fit of the camera model to the measured data did not converge. Existing software was modified to allow for the manual editing of the initial camera model estimates. Editing was accomplished with a graphic display showing the measured calibration points and calibration images. The elimination of outlying calibration points and the selection of a good initial estimate allowed the camera models to converge.

POSSIBLE IMPACTS ON SPACE FLIGHT AND SPACE STATION

Modifications in Requirements

In order to provide for the increased activity of ground/remote telerobotic operations, modifications will need to be made which provide the appropriate level of communication with Earth. Modifications of the FTS for proximity sensors, increased video coverage, and required local processing should also be made, e.g. for reflex actions. An Operator Control Station and processing facilities on Earth would also be required as part of the Space Station design. To make the operations more amenable to robotic manipulation, tools and jigs should be designed.

Benefits

The benefits of ground/remote telerobotics for the space station include a 24 hour/day work cycle for Space Station assembly with alternating ground personnel controlling the assembly robots, and improved astronaut safety through reduced EVA. Reliability is also improved by eliminating repetitive, menial, and tiring tasks from the operator's work-load.

FUTURE PLANS

There are many plans for future work, as time and budget allow. The following are a sample of the items which will be incorporated into the present system at a future time.

Kennedy Space Center

1. Development of requirements and design proximity sensors for the IRB90.

2. Design, build and integrate a two degree-of-freedom articulated "boom" extension to the IRB90.

3. Design, build and integrate a video camera system for the extended IRB90.

4. Develop an accurate kinematic model of the extended IRB90.

5. Install the TPR subsystem software on a artificial intelligence workstation located at KSC.

Jet Propulsion Laboratory

1. Expansion of the IGES world model to allow for more flexible operations, and more viewpoints. This requires the addition of a fine-motion spatial planner.

2. Improvement of the operator interface and overall system speed.

3. Addition of fine-motion spatial planning to enable the IRB90 to move to arbitrary positions.

4. Development of models for a modified IRB90 and a new camera system.

5. Transition of the machine vision system to a next-generation VME-based hardware platform.

FUTURE CHALLENGES

The JPL/KSC team faces some future challenges which can be met by a well-designed research effort.

Proximity Sensing

The design of the proximity sensors should aid in increasing safety, while the information from the sensors should be utilized for spatial planning.

Spatial Planning

1. Improvement of gross-motion spatial planning by speeding the graph generation.

2. Integration of fine-motion with gross-motion spatial planning.

3. Development of spatial planning for an incompletely or erroneously modeled environment.

4. Integration of spatial planning tools with operator interface to resolve spatial

problems and to make the spatial planning faster and more reliable.

Perception

1. Localization of objects when *a priori* location is unknown.

2. Characterization of known objects.

3. Effective modeling of a complex environment.

ACKNOWLEDGEMENTS

The research described in this publication was carried out jointly by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration, and by the Kennedy Space Center, National Aeronautics and Space Administration.

We gratefully acknowledge the support of V. Leon Davis, Sven Grenander, Bert Hansen III, Bob Lewis, Wayne Shober, and Brian Wilcox in the administration of this research.

Requests for reprints should be sent to David Mittman, Mail Stop 301-250D, Jet Propulsion Laboratory, Pasadena, California 91109.

REFERENCES

Bon, B. (Ed.). (1989, November). JPL-KSC telerobotic inspection and manipulation demonstration: FY 1989 final report. (Available from David Mittman, Jet Propulsion Laboratory, 4800 Oak Grove Drive, Pasadena, California, 91109, Mail Stop 301-250D)

Cain, R. (1987). NIP user's guide. Menlo Park, California: SRI International.

Collins, C. & Rokey, M. (1988). Planning for the Jet Propulsion Laboratory telerobotics project. In Proceedings of the Artificial Intelligence & Advanced Computer Technology Conference (pp. 429-437). Glen Ellyn, IL: Tower Conference Management.

Gennery, D. B., Litwin, T., Wilcox, B., & Bon, B. (1987). Sensing and perception research for space telerobotics at JPL. In Proceedings of the IEEE International Conference on Robotics and Automation (pp. 311-317). Raleigh, North Carolina: IEEE.

Hernson, J. (1987). NIP user's guide: Symbolics LISP machine supplement. Menlo Park, California: SRI International.

Jet Propulsion Laboratory. (1989). Code ST advanced development task description: Architecture for telerobotic systems. (UPN 476-86-03). (Available from David Mittman, Jet Propulsion Laboratory, 4800 Oak Grove Drive, Pasadena, California, 91109, Mail Stop 301-250D)

Kennedy Space Center. (1978). Launch site accommodations handbook for STS payloads. (NASA-TM-80419). John F. Kennedy Space Center, Florida: National Aeronautics and Space Administration. (NTIS No. N79-76877).

Mittman, D. (1988). Audrey: An interactive simulation and spatial planning environment for the NASA telerobot system. In Proceedings of the Artificial Intelligence & Advanced Computer Technology Conference (pp. 421-428). Glen Ellyn, IL: Tower Conference Management.

Peters, S. F. (1988). Autonomy through interaction: The JPL telerobot testbed interactive planning system. In Proceedings of the SPIE Conference on Space Station Automation. Boston: SPIE.

Peters, S. F. (in press). Planning and reasoning. In S. Tzafestas (Ed.), Intelligent robotic systems: Analysis, design, and programming. New York: Marcel Dekker.

Peters, S. F., Collins, C., Mittman, D., O'Meara, J., & Rokey, M. (in press). Planning and reasoning in the JPL telerobot testbed. Pasadena, California: Jet Propulsion Laboratory.

Yakimovsky, Y., & Cunningham, R. (1978). A system for extracting three-dimensional measurements from a stereo pair of TV cameras. Computer graphics and image processing, 7, 195-210

N90-22313

SYSTEM CONTROL OF AN AUTONOMOUS PLANETARY MOBILE SPACECRAFT

William C. Dias
Barbara A. Zimmerman
Sequence Automation Research Group
Mission Profile and Sequencing Section
Jet Propulsion Laboratory
California Institute of Technology

PHONE: Dias (818) 354-0153 Zimmerman (818) 354-6700
MAIL: Jet Propulsion Laboratory, MS 301-250D
4800 Oak Grove Drive, Pasadena, CA 91109

ABSTRACT

Our goal is to suggest the scheduling and control functions necessary for accomplishing mission objectives of a fairly autonomous interplanetary mobile spacecraft, while maximizing reliability. Goals are (a) to provide an extensible, reliable system conservative in its use of on-board resources, while (b) getting full value from subsystem autonomy, and (c) avoiding the lure of ground micromanagement. We propose a functional layout consisting of four basic elements: GROUND and SYSTEM EXECUTIVE system functions and RESOURCE CONTROL and ACTIVITY MANAGER subsystem functions. The system executive includes six subfunctions: SYSTEM MANAGER, SYSTEM FAULT PROTECTION, PLANNER, SCHEDULE ADAPTER, EVENT MONITOR and RESOURCE MONITOR. The full configuration is needed for autonomous operation on Moon or Mars, whereas a reduced version without the planning, schedule adaption and event monitoring functions could be appropriate for lower-autonomy use on the Moon. An implementation concept is suggested which is conservative in use of system resources and consists of modules combined with a network communications fabric. The paper introduces a language concept we have termed a "scheduling calculus" for rapidly performing essential on-board schedule adaption functions.

INTRODUCTION

Interplanetary mobile spacecraft (rovers) require more autonomy than spacecraft in planetary flybys or orbiters, if they are to be acceptably productive. This is essentially because knowledge of the environment changes over a much shorter time scale than the speed at which data could be received and analyzed and commands generated and sent from Earth, given the light-time delays (Wilcox, et al, 1987; Dias, et al, 1987). Even a low autonomy rover on the relatively nearby moon needs more autonomy in the control area than other spacecraft if it is required to move continuously (Pivirotto, et al, 1989).

This paper proposes a FUNCTIONAL SYSTEM CONTROL ARCHITECTURE in which a design or requirements for a design could be phrased. We address fairly autonomous rovers first of all. Second, adaption of the control architecture to a low-autonomy Lunar rover is discussed. Next, the paper has a section on the practicalities of implementing the control architecture. Last, we discuss ongoing research in the JPL Sequence Automation Research Group on the development of a language in which the rule base of vital parts of the control system could be phrased.

The design process, as well as the design itself, should be responsive to the needs of operations managers to ascertain reliability and functionality. This is because the control system partly substitutes functionally for ground operations. Fairly autonomous rovers would need to be able to reliably perform many of the spacecraft command and control functions now performed only on Earth (Linick, 1985). It will be seen the functional layout preserves some of the current division of responsibilities among traditional ground system and subsystem elements. Various parts of the COMMAND GENERATION process, including REQUEST GENERATION, REQUEST INTEGRATION, SCHEDULE GENERATION and COMMAND TRANSMISSION, are proposed for on-board implementation.

Our proposed architecture assumes a spacecraft with a complex and varied set of goals and activities only partly predictable during design. Activity schedules will require some parallelism and optimization, as now provided for on Voyager and Galileo. There will inevitably be a desire for a great degree of ground control, to maximize mission return. In fact the command and control system design must walk a tightrope among the three paradoxically competing concepts of system autonomy, subsystem autonomy, and maximal ground control, each proffered in the name of maximizing return.

The control system needs to be as VERSATILE as possible, because the exact desired operational modes and combinations of activities for a spacecraft are not always fully knowable in advance, and this will be especially so for a planetary rover. The less known in advance about the particulars of the environment, the more varied that environment, the more varied and general the set of tasks, and the larger the suite of approved instruments, the less predictable the final operational range will be.

To promote rover functional EXTENSIBILITY, the control system design should incorporate features able to enhance the software development environment. Quick changes may be needed in the software implementing traverse and sample acquisition functions after landing, whether due to unforseeable hardware failures or unexpected conditions. Depending of course on the mission design, sample return mission surface stay times could be as short as a few months, adding greatly to pressures for operational responsiveness (Bourke, et al, 1989). Rapid software turnaround presents a danger of its own in an operational environment. By being versatile and robust enough to comprehensively trap and correct fault conditions, a good control system design should make fast software development turnaround possible in the operational phase. We can define the architecture to maximize probability of success. We have done this by incorporating software verification in the fault protection scheme.

Our architecture differs considerably from other proposals, though it takes a layered approach often favored by other designers (IKI, 1988; Simmons, et al, 1989). Resulting as it does from the considerations in the above paragraphs, our proposal is oriented towards providing "general purpose" spacecraft functionality by representing what currently exists as ground operations functions on board. This approach differs from Subsumption Architecture (Brooks, et al, 1989) and from the Task Control Architecture (Simmons, et al, 1989). These appear to be oriented towards a predefined (but presumably robust) set of "behaviors", and towards missions which could be accomplished with rovers operating in a more narrowly defined functional envelope than the kind of mission we foresee. We feel early interplanetary rover missions will need to use mobile spacecraft which are as general in capability as is reasonable, for all the reasons in the above paragraphs. It seems likely there would be a place for both the "behavioral" and "general purpose" architectural philosophies in a well funded, longer term solar system planetary exploration program, however.

The functions required for the rover as a whole are taken generally from Smith and Matijevic (Smith, et al, 1989). We have added Global Navigation, Data Handling, and Pointing and Articulation to their System Executive, Telecommunications, Power, Thermal, Science, Mobility and Sampling. Thus, our idea on how these subsystem functions should be defined is slightly different, but exact subsystem delineations are not our purpose. Instead, our hope is to clarify the system / subsystem interface in general. We find no essential conflict between the Smith and Matijevic formulation and ours. The emphasis is different. Their method appears to provide a convenient means for designating the control, command, and data paths to be included in a roving spacecraft from high to low levels, before design begins. Where they provide a general purpose tool and framework, we try to provide and justify the functional relationships which need to be used to fill in the details in the Smith and Matijevic architecture matrix, with emphasis on the System Executive over the subsystems. We wanted to show the most meaningful functional interfaces and formulate them so that people can begin to think of allocation of functions to modules.

The control system needs to incorporate concepts from spacecraft FAULT PROTECTION (Riethle, 1983) in order to improve RELIABILITY over research and ground-based robots and robotic vehicles. In a "classic" form of fault protection, signals from one or a few subsystems are used to determine a fault and then pre-canned, simple and highly structured routines take control of the subsystem or spacecraft and throw it into a predetermined, safe, but usually non-productive state. This "classic" form of fault protection needs to continue to exist on planetary rovers, but its scope needs to be limited in such a way that more intelligent autonomy is not subverted by its sheer simple-mindedness. More refined forms of fault protection which take into account the higher level of intelligence on the spacecraft must be included, or the advantages of intelligent autonomy would be lost.

Finally, it may seem that this architecture is too complex, that it could never execute in a timely fashion. We do not take this potentially serious problem lightly. The functional description appears complex partly because we did not want it to appear incomplete through overgeneralization. We wanted to try to bring out a description of the functions and interrelationships that might be required, perhaps more complete than available in the past. The design and implementation could turn out considerably simpler than the functional description might cause one to expect, but all functions should be addressed in the design process. We also feel we have allayed some of these complexity concerns in the implementation section.

OVERVIEW

This section contains an abbreviated description of the overall control flow of the architecture. In the subsequent detailed section on selected elements, we give a fuller description of the behavior of the major elements.

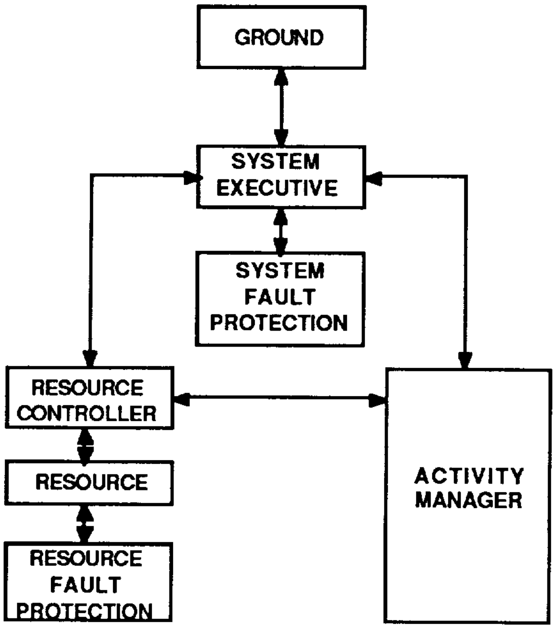


Figure 1. Basic Functional Control Architecture Layout.

The functional control architecture has four basic elements: GROUND, SYSTEM EXECUTIVE (SE), ACTIVITY MANAGERS (AMs), and RESOURCE CONTROLLERS (RCs). These share control as shown in Figure 1. Ground and SE in combination provide the system-level command and control functions, while the subsystem functions are performed by the AMs and RCs in various combinations for different operational modes. The SE and RCs each have separate fault protection functions, but the AMs are oriented more strictly towards command generation, command, and control, and have no fault protection role.

In the operational scenario for autonomous modes, which are the ones addressed in this paper, GROUND provides goals and schedule contraints -- general and specific -- to the rover SE. GROUND uses some sort of simulation based on whatever information it has from orbit, previous traverses, statistical likelihoods, and the rover's sensors to try to foresee likelihood of success. There is no guarantee that goals are achievable within time constraints, only some probability. Goals should try to encompass at least a few hours of activity, preferably a day or so, with fallback contingencies in case operations take longer than expected, and fill-in items in case activities take less time than foreseen.

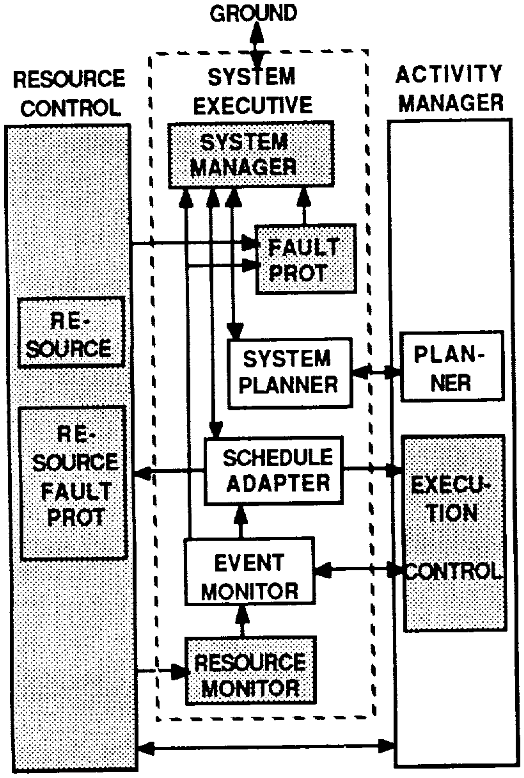


Figure 2. System Executive command/control architecture. All functions required on more autonomous rovers. Only shaded areas needed for command/control of low-autonomy rovers.

Figure 2 depicts the internal functional interfaces of the System Executive in more detail. The on-board SE co-ordinates rover operations through the SYSTEM MANAGER. First the SYSTEM PLANNER and the requisite subsystem AM PLANNERS agree on a plan to accomplish the goals. They do this by first having the SYSTEM PLANNER arrive at sub-goal and activity sequences and resource and time envelopes within which the AMs must plan, then having the AM PLANNERs arrive at sequences implementing the goals to the degree possible. A series of "events" would be part of any schedule agreed upon. These events would be more or less at the same level as those used by a Voyager or Galileo ground sequence team in its higher level scheduling activities today. The time relationships among these would be partly relative, that is, the time at which an event would occur would be phrased as relative to other events, not as an absolute time (though the system would always have a nominal absolute time for future events.) Sequences are phrased relative to events, with the exact time of commanding to be determined later, in execution.

In the execution phase of the plan, the AM EXECUTION CONTROLLER actually sends commands to (and reads data from) the RESOURCE CONTROLLERS, which are the only entities able to address the physical resources directly. The RESOURCE MONITOR (RM) keeps track of resource status (including progress on events) at the system level, by reports and / or polling techniques. The EVENT MONITOR keeps tabs on the progress of the schedule with respect to the events. It does this by receiving both timed and event-tagged progress reports from the AMs, and (redundantly) by RC reports relayed from the RM. Discrepancies in these reports result in fault protection, replanning, or other exception processing. AMs may optionally read event reports from the EVENT MONITOR. The SCHEDULE ADAPTER constantly modifies the timing of the system-level sequence, within agreed parameters, based on event-progress reports from the EVENT MONITOR. The AMs modify their schedules for data and for commanding the RCs based on near-realtime scheduling refinements from the SCHEDULE ADAPTER. The latter is also able to dynamically reconfigure the RCs based on schedule changes, and the AMs are then limited to commanding within those parameters.

Replanning can occur during execution, either because specific events in a schedule are designated as points for plan refinement, when the next increment of information is available for planning, or because unforeseen conditions caused an existing schedule to be invalidated. Sometimes, subsystem replanning will be necessary without system replanning being needed.

RESOURCE FAULT PROTECTION responses, including reflexes, may be invoked to safe a resource based on resource-internal information alone. Signals are then sent system-wide. SYSTEM FAULT PROTECTION may be invoked in response to conditions signaled by combinations of subsystems, the schedule becoming dangerously unworkable, disagreeing progress reports from the AMs and RCs, or signaling of computed status derivatives and trends indicative of faults from the RM. One would try to autonomously replan out of at least some fault protection response modes.

DETAILED FUNCTIONS OF SELECTED SYSTEM ELEMENTS

This section discusses selected elements of the architecture in more detail than in the overview.

SYSTEM PLANNER

First, the System Planner processes ground-supplied GOALS into a serial and parallel collection of MAJOR ACTIVITIES which will bring about the desired goals. Time factors are only applied in a gross way, which is enough to eliminate many schedule possibilities.

Second, the System Planner derives schedule constraints and resource utilization envelopes within which each major activity must be planned.

Next, the System Planner waits while subsystem AM Planner functions derive a SCHEDULE from the required activities (See Activity Managers, below). The System Planner then integrates the resultant subsystem-derived schedules, which may include changing the previously imposed resource and schedule constraints and asking for additional subsystem planning.

The SCHEDULE derived by the planning functions must have a form consonant with the need for on-board schedule maintenance in realtime. This is more than is required of a contemporary spacecraft schedule. In general a schedule might be defined as the timed series of events and states of all resources and subsystems which is determined to bring the desired activities to completion. In order to allow for later realtime adaption, the new type of schedule must include temporal relations among events, of a sufficiently economical nature to allow operations in realtime. In other words, an EVENT-DRIVEN SCHEDULE is needed. Each event or state change needs to have its time requirements described relative to when other events or state changes take place.

Stated more abstractly, the schedule consists of a set of functional relationships among the three elements of states, events and times, such that, where *t* is an instant in time and F1 and F2 are functions:

1. System State$_t$ = F1(Subsystem States$_t$)
2. Acceptable System State$_t$ = F2(System State$_{t-1}$)

and, where F3 is a function, Event$_a$ is next on the schedule and Events *b1* through *bx* have already occurred:

3. Acceptable Time Interval(Event$_a$) = F3(Times (Event$_{b1}$,...,Event$_{bx}$))

Applied recursively, what this expresses is that acceptable activities or states of both the system and of each subsystem, for any instant in time, depend on the states of all those entities dynamically as matters progress. This is similar to a system which runs according to "control laws". In this way, schedules are derived without assigning all final times, but with needed time linkage among events and states. In the last section in the paper, we discuss a language and some of the rules of a "scheduling calculus" which could be used to express and enforce the functions in a real system.

The following are possible examples of "events" at the level of interest of the System Planner. These correspond roughly to a fairly high level of planning, specifically the initial sequence integration which occurs right after request generation for a spacecraft. They stop short of device management which in this conception is at the level of the subsystem AM Planner and RC.

start / finish a camera platform slewing operation
start / finish a single maneuver in the course of a longer traverse
start / finish a sample arm movement (or, maybe a joint movement)
start / finish warmup of a sample processing oven

SCHEDULE ADAPTER

The Schedule Adapter converts the schedule, phrased in expressions to be operated on by the scheduling calculus, into a high-level SEQUENCE with final times, exact states, etc., assigned. Whereas "schedules" include functional relationships among times, events and states, "sequences" include only the times, events and states which result from applying the functions based on knowledge at a given instant. Thus what is known about the spacecraft high level sequence might include only a few seconds or minutes of activity.

So the Schedule Adapter routinely uses the rules of the scheduling calculus to derive acceptable ranges for the next set of all subsystem states from the current state. In a closely related function, it continually changes the system's idea of when events will occur. These adaptions thus do not necessarily constitute a need for schedule changes (i.e., replanning) in our terminology.

The Schedule Adapter is very dependent on input from the Event Monitor to keep it informed of status of all relevant events, or uncertainty in that status. The Schedule Adapter must have a way to respond to unclear state or event status knowledge, informing System Fault Protection which may be the one to decide what to do.

The Schedule Adapter sends sequence revisions to the relevant AM execution control functions, which adapt in turn.

The Schedule Adapter provides configuration commands to the RCs. The AMs, which actually run the rover through an activity such as a rolling maneuver, are limited to commanding the RCs within the envelopes configured by the Schedule Adapter for each moment in time. This mechanism provides system level resource control while allowing the responsible AMs reasonable command authority over those resources needed.

There may have been specific points in the schedule designated for system or subsystem replanning or plan refinement. If so, these are honored in an event-driven fashion the same as other dependent events.

The Schedule Adapter may be called upon by System Fault Protection to invoke a special schedule implementing a fault protection schedule, and to abort a current schedule in an organized way.

Another part of the Schedule Adapter's function is to realize when incompatible combinations of conditions occur or are about to occur. For instance, if the rover is running behind schedule in getting to a site of scientific interest, low priority activities may need to be either rushed through, with controlled loss of data quality, or abandoned entirely. It is up to the Schedule Adapter to do this and signal the AMs and RCs accordingly. If conditions deteriorate further, the System Planner and AM Planners must be reinvoked to completely rework the schedule and salvage what is possible of the original goals. As a last resort, the rover should inform Earth of the problem at the next opportunity. Ground can then respond by recommanding with adjusted goals. This can be an expensive solution in terms of wasted rover time, a fact that needs to be worked into the original decision on what to do if a schedule fails under various conditions.

SYSTEM EVENT MONITOR

This function holds the current system knowledge of all events previously completed/aborted in the schedule or in progress. It accomplishes this by seperately monitoring both resource and activity status. This redundant approach provides cross-checking considered highly desirable in spacecraft fault monitoring (Riethle, 1983; Reiners, 1985).

First, AM Execution Controllers provide activity status to the Event Monitor, both at predetermined intervals and at status change points agreed to in the schedule. This is a high level check that activities are on schedule and status is acceptable. Activity status report frequency will undoubtedly vary by operational mode. Events to be reported can be unplanned. For instance when the AM Execution Controller becomes aware independently that its plan is no longer workable, that needs to be reported.

Second, event-related resource status reports are provided from the System Resource Monitor, which has collected these from the RCs. Unplanned events, such as the invocation of a reflex action by the RFPCs, also need to be reported.

The Event Monitor integrates these various sources of event status and reports to the Schedule Adapter to help it make decisions. System Fault Protection is informed in case event patterns reported can be used to infer fault conditions. The software validation function is served because some (perhaps most) software problems will show up as error reports or as inconsistencies in event reports among the various sources.

SYSTEM FAULT PROTECTION (SFP)

This function includes the separate areas of fault detection, fault analysis, and fault reponse. It detects system-level fault conditions from combined messages from the System Event Monitor, System Resource Monitor, Resource Fault Protection Controller, and Resource Controllers. Messages can include both status and data determined in the design process. It may execute hard-coded (or at least high-speed), high-reliability responses to faults detected, forcing the spacecraft into very well defined states from which recovery will be as easy as possible. It may conceivably also initiate slower fault responses requiring normal schedule planning channels. It seems likely System Fault Protection would include reliable, predesigned, canned schedules in a form able to be adapted to specific current conditions by the on-board Planners and/or Schedule Adapter.

The System Fault Protection functions need to respond directly from primary inputs -- otherwise they would be dependent on other functions and therefore less foolproof. At least some responses must be designed to operate without permission from the System Manager. The System Manager must in turn be informed as soon as fault protection is invoked. This requirement poses a problem faced by all systems with distributed authority and / or redundant data -- the possibility that different parts of the system will be working to cross purposes for some interval of time, with resultant system state ambiguities. The problem cannot be fully addressed until the design phase. Hopefully, the pre-canned fault protection schedules can be designed so as to be adaptable by the Schedule Adapter to the particulars of the current state.

SYSTEM RESOURCE MONITOR

The System Resource Monitor has three separate ways of monitoring resource status.

First, the SE receives a "heartbeat" -- or elementary status message -- from each RC on a regular, timed basis. These messages are independent of any specific task the resource has been commanded to perform. The total absence of a message, a message conveying an error status, or a message containing data from which an error condition is deduced, can be grounds to invoke system-level fault response. Heartbeat occurence and frequency may vary from subsystem to subsystem, but as a point of reference, Galileo heartbeats are at approximately 0.7-second intervals.

Second, the RCs report to the Resource Monitor in connection with the specific tasks they have been commanded to support. These messages are tagged to the portion of the command sequence that brought them about. A "resource event" in this sense includes any RC status change, or any change in the tasks being supported even though there may be no other resource status change. Reports are also required at regular intervals (much like the heartbeats) as a cross-check that status is as expected and things are on schedule.

Third, signals from the RFPC are received in the event resource-level fault protection is invoked.

Both planned and unplanned resource status changes are events and are duly reported to the System Event Monitor.

ACTIVITY MANAGERS (AMs)

Identified subsystem functions requiring AMs are: Science Payload, Sample Acquisition, Traverse, Global Navigation, Telecom, Data Handling, and Imaging. We discuss the AMs only to the degree necessary to put them in context with the rest of the rover control system. Their design is specialized, different for each subsystem function, and not the subject of this paper.

AMs provide a level of intelligence higher than the RCs for important spacecraft subfunctions (e.g., autonomous power subsystem, Fesq, et al, 1989; autonomous navigation subsystem, Gat, et al, 1989). They trade or share control depending on operational mode. On less autonomous spacecraft, subsystem planning and monitoring would be performed by ground subsystem engineers or scientists in coordination with system level engineers. On a more autonomous spacecraft such as a fairly autonomous rover, the AMs to some extent represent on-board the functions the subsystem engineers serve on the ground. AMs have NO FAULT PROTECTION RESPONSIBILITIES, because it would be redundant, and because they represent areas such as navigation where fast software development turnaround is desirable. AMs requiring data from other AMs for planning or execution functions must obtain and update information in common data base areas to maintain controls.

The fully implemented AM is presumed to have a PLANNER and an EXECUTION CONTROLLER.

The AM PLANNER utilizes specialized subsystem knowledge unavailable to the SE planner to derive (1) a sequence of time-driven and event-driven commands to be given to the RCs, and (2) a corresponding set of expectations (Gat, et al, 1989). The AM Planner co-ordinates provisional plans with the SE Planner. It should be noted that, as in the case of the SE planner, planning may be incremental. That is, a high level activity such as the acquisition of a sample will probably be worked out as a series of steps with estimated times, with final planning applied only as preceding steps complete.

The AM EXECUTION CONTROLLER co-ordinates execution of plans agreed to between the AM Planner and the SE Planner. It implements control by commanding the RCs and reading data and status from them. It may read event status, if desired, from the SE Event Monitor. It responds to schedule envelope changes provided by the Schedule Adapter in near realtime. It reports its version of events and status back to the Event Monitor on both a time- and event-driven basis. It may be interrupted by the SE if the latter decides things are not on track and invokes fault protection or replanning.

RESOURCE CONTROLLERS (RCs)

Every resource is governed by a Resource Controller (RC) which has about the same level of intelligence as a typical contemporary disk controller. All contact between a resource and other elements (except resource-level fault protection) is through its RC. RCs for different purposes could have different amounts of memory and CPU power, but they would all have the same qualitative functions: receiving commands, sending status, sending and receiving data, and keeping track of a time-linked stack of commands for one resource. The RC accepts reconfiguration commands from the Schedule Adapter and commands from the AM Execution Controllers, provided these are within the configuration envelope provided by the Schedule Adapter. It provides both time- and event-tagged status to the System Resource Monitor and System Fault Protection functions. Status messages returned by the RC include identifiers so that other functions may know which event(s) from the sequence the resource is currently working on, and the RC's progress on the sequence.

RESOURCE FAULT PROTECTION CONTROLLERS (RFPCs)

Any precanned, fixed routines which are designed to automatically, unconditionally and unilaterally change individual resource states based on sensor fault readings are handled by the RFPCs. This is a basic, conventional spacecraft fault protection strategy which will continue to be needed for some faults. Examples include fuse protection, automatic shutdown of electric heaters exceeding temperature specs, or trend analysis for individual resources. Other system elements such as the RCs, AMs and SE Event Monitor are informed of the fault status through normal channels after the fact.

In some cases, a system-wide fault response is needed, which must be processed by System Fault Protection (SFP). In those predefined cases, the duty of the RFPC is to simply send the status to the RC and the important fault data to the SFP for action.

In our view, RFPCs would also implement any required resource-level reflexes. These include any action or behavior, at the level of the individual resource, required on an unexpected basis and on too short notice for organized involvement by the planning functions. We believe reflexes should be considered fault reponses for spacecraft design purposes. Of course, not all reflexes result from true emergencies and will therefore sometimes result in situations easily handled by on-board software.

The invocation of any reflexes or resource-level fault protection presumably necessitates replanning after any further immediate spacecraft safing is complete.

RESOURCES

Resources are commandable elements providing services, conditions, or commodities to the requesting elements, through their RCs. All communication to a resource is through the RC in normal modes. The following commandable resources have been identified for a planetary rover: Power, Thermal State, Data Handling, Science Payload, Science Imaging, Sampling Mechanisms, Pointing and Articulation, Mobility / Vehicle State, Mass Storage, and Telecom Data.

At this stage of the design, there is always the possibility that some high-speed control requirement will later be found requiring direct communication with the resource. That discovery will have to await the testbed development stage.

LOW-AUTONOMY INTERPLANETARY ROVERS

True teleoperation, in which both command and low-level control is in the hands of an operator, is thought to be an unreasonable means of controlling rovers even on the Moon. The light time delay of around three seconds is too great (Pivirotto, et al, 1989). 300 milliseconds may be the maximum for teleoperation.

Our view is that all the functional elements discussed in this paper would be needed in some degree for fairly autonomous interplanetary rovers, regardless of the distance from Earth. However, considerable economy is possible for a Lunar rover with lower autonomy and interactive commanding from Earth, because the round-trip communication delay (light-time plus electronic delays) is likely to be only a few seconds. All planning (system and subsystem), schedule maintenance and event monitoring functions can be done on Earth. These would be tightly integrated in operator command terminals. Activity execution monitoring, mode switching, resource handling and fault protection would be on the spacecraft. See shaded areas on Figure 1. This is more autonomy and on-board control than would be provided by teleoperation.

It is likely that Lunar rover programs will naturally precede Mars rovers (*Report of the 90-Day Study*, 1989). This provides an opportunity to perfect the designs and techniques for autonomous schedule maintenance on the ground in an earlier program. Those functions would later be moved on-board to achieve the greater autonomy necessary for productivity at the up to 40-minute round trip light time delays presented by Mars, or by more autonomous Lunar rovers.

IMPLEMENTATION OF THE SYSTEM EXECUTIVE ARCHITECTURE

To the SE the planetary-vehicle domain appears to be made up of a two-level hierarchy of elements. The top level of the hierarchy represents the vehicle functions or activities. The lower represents the hardware that participates in carrying out the activities. This view of the rover domain suggests an architecture that supports multiple interacting tasks which can access a pool of resources. A suitable architecture can be modeled, superficially, by a modern computer operating system (Rashid, 1986). However, the computer operating system model is not complete or sufficient because the control procedures commonly invoked fall short of those required for a capable rover's operation. To control a fairly autonomous planetary vehicle the system design must specify a comprehensive self-analysis tool with which to track the state of the vehicle. In this section we describe an architecture for the SE's control functions, and a procedure for planning and diagnosis of the state of the rover as it carries out a task. The procedure, which we call a scheduling calculus, combines qualitative relationships with arithmetic expressions to render judgements about the rover's state, and the validity of a schedule or sequence given that state.

OVERVIEW

Preliminary system designs (Pivirotto, et al, 1989; Lambert, 1989) have typical planetary vehicle functions such as sample acquisition, navigation, data handling, science, imaging, and telecommunications controlled by several independent processing elements (See Figure 3).

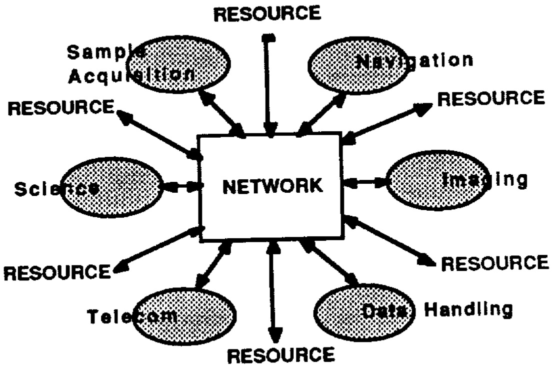


Figure 3. A Distributed Computing Network. Processing Elements (ovals) and Resources controlled by independent activity modules through a common system-level network.

In addition, the processing elements share some form of mass storage and system resources. The processors and the resources are joined by a communication fabric which connects each of the processing elements to one another and to the resources. We will refer to this communication fabric as a network. We envision a layered architecture whereby the activities direct their requests for resource usage through the SE (Figure 4). Access to the SE, the activities, and the resources is by message passing which is supported by SE service routines.

An important concept in the proposed design is that the SE's subfunction modules are modelled as a collection of one or more independent processes that communicate through message passing. The modular nature permits the SE to be dynamically configurable to accomodate the requirements of a mission. Further, we envision a system in which one or more of the SE's subfunction processes reside in each processing element. The distributed capability, which is independent of the number of processors, can be used to partition the demand on the rover's computer resources.

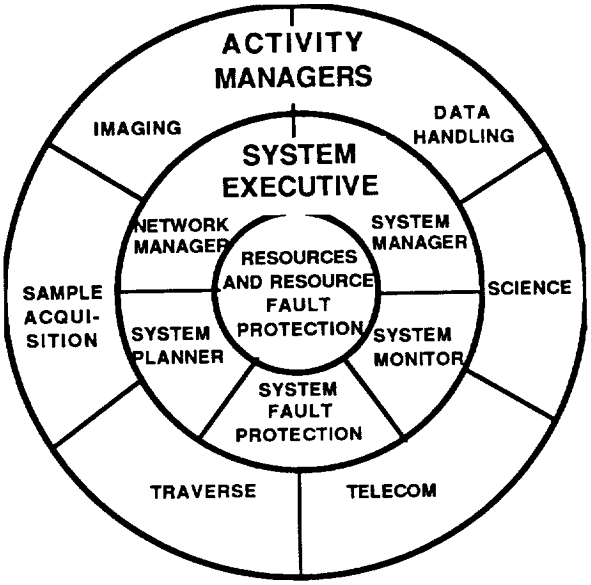


Figure 4. Rover Command / Control System Layered Architectural Layout.

SYSTEM EXECUTIVE MODULES

The following sections describe the five classes of modules that make up the SE. These are the SYSTEM MANAGER, SYSTEM FAULT PROTECTION, SYSTEM PLANNER, SYSTEM MONITOR, and SYSTEM NETWORK MANAGER. It is assumed that the system executive processes reside on top of some form of computer operating system environment which provides the low-level functionality with which the hardware is addressed and the data managed. In a distributed system there has to be some means by which data of a global nature are provided to the processes. While the SE, the RCs and the AMs depend on these services, they are not part of the SE functions and are not discussed in what follows.

System Manager

This element of the SE provides the interface between the ground and vehicle, and among some of the major on-board components. The system manager configures the control system to support the mission requirements. For a less autonomous rover that is to be commanded from the ground, the system manager directs the command sequence to the relevant AM Execution Controllers and Resource Controllers. For a more autonomous rover the System Manager activates the planner processes, and the event monitor.

System Fault Protection

Both lower- and higher-autonomy rovers need on-board fault protection. It is likely the great body of fault detection, analysis and response code could be thought of as stored in this separate class of modules. However, it is necessary for efficiency to distribute some of the fault detection responsibilities to the System Monitor and Network Manager modules, outlined below. In addition some elementary fault responses, somewhat redundant, will probably be required as part of the code running in each node in a multiprocessor network. As stated elsewhere, other fault protection is implemented at the subsystem level, entirely outside the SE. However, it is the responsibility of the system fault protection class of modules, in the event of a system fault, to activate fault protection, retrieve the necessary data to configure the resources and return the rover to a known state, and send the necessary commands to orchestrate controlled aborts of on-going processes. In the higher-autonomy rover, stored, configurable schedules can be provided on-board to the System Planner and Schedule Adapter for these purposes.

System Planner

The two planner processes, a Planner and a Schedule Adapter, provide a layered implementation of the scheduling processes. The system manager directs relatively high level goal and constraint data to the system planner. The goals are expanded by the planner into a time-ordered set of tasks for the Activity Managers to process. The Activity Managers produce the detailed sequence of events that accomplish the tasks. Before execution, this sequence is returned to the planner for verification and Event Token extraction (see below). The schedule adapter monitors the plan's execution, adjusting the sequence as needed.

There are several features that reflect the degree of sophistication represented by the SE planner and schedule adapter processes. The planner must automatically generate and maintain a schedule. This is a significant task which ground-based planners currently do not fully support. We are investigating the Remote Mission Specialist (RMS) system developed by the Sequence Automation Research Group at JPL (Rokey, et al, 1990), and the research on automatic planning by the same group (Eggemeyer, et al, 1990) for solutions to the planner requirements. In addition, our own work on the scheduling calculus can be used for the schedule maintenance task.

System Monitor

The system monitor provides for the required event and resource monitoring functions. This includes maintaining current resource status for system purposes, and serving as a central repository of event status around which other modules can synchronize. It is expected that some of the fault detection -- or preprocessing for that purpose, such as trend analysis -- would be offloaded from the system fault protection modules to the monitor modules, for efficiency.

Network Manager

All of the rover elements, including the SE, communicate via message passing. The network manager provides the system services to support this activity. The system services provide a message format that characterizes the nature of the communication. The message format includes fields for the message type, the sender, the event tag, and the command or data, and possibly the destination. Based upon message type, the messages are expeditiously forwarded to relevant processes. The sender of a message need not know how a request for data will be filled, or how a message will be routed. For example if the message type is that of a "heartbeat" message from a Resource Controller it is directed by the network manager to the Resource Monitor process of the SE.

The network manager provides for both added security and ease of application programming. The subsystem implementers access all of the rover functions through a uniform interface -- a message protocol. In addition, the network manager implicitly provides the capability to monitor message traffic over the network. Statistics gathered by this process can be used to measure the health of the subsystems on the net. However, care must be taken in the design of this type of support to insure that the service does not overwhelm the computer resources.

SCHEDULING CALCULUS

The overall function of the schedule adapter and monitor processes is to integrate the responses from multiple task activities to determine whether the activities are leading to the given objectives or to an undesirable state or possible fault condition. We are required to assign a qualitative value for the state of the vehicle, based upon quantitative information supplied by the resources. In addition, we are required to judge whether the events reported during the execution of a sequence match the goal set for the task, or whether conditions of the vehicle or in the environment require the sequence to be changed. We chose to investigate the methods of qualitative processing (Bobrow, 1985) as an approach to solving the above problem. In particular, we are in the process of developing a language that implements an arithmetic reasoning tool. The work follows closely that described in the paper *Commonsense Arithmetic Reasoning* (Simmons, 1986). We are using the UNIX[1] yacc procedure (Johnson, 1983) to build the scheduling calculus language. The language combines methods of graph search, interval arithmetic, relational arithmetic, constraint propagation, and function evaluation to arrive at decisions about the validity of a schedule step, and conclusions about the rover state. A capability of the

[1] UNIX is a trademark of AT&T.

language can be illustrated by one of Simmons' examples. Given the ranges of three quantities A, B and C we want to determine the acceptable range for a fourth quantity, D:

A is constrained to the interval [3,4]
B is constrained to the interval [1,4]
C always has the value of 2
D is related to A,B,C by (B*C)/(A+B)
We assert the relation B >= C
B is now constrained to interval [2,4]
Using interval arithmetic we determine D
    is constrained to [.5,1.6]

REQUIRED KNOWLEDGE BASE

The tool will require a knowledge base that provides functions, parameters, values, and constraints that describe an acceptable operating environment. We call this a model database. The high-level goals provided by the uplink process will provide further functions and constraints with which to reason about the system. The event tokens derived by the planning process and resource sensor data also contribute to the knowledge base. We call this the derived knowledge base. Using the model and derived bases, directed graphs can be built by the language as reasoning tools. Values for the functions are the nodes of the graphs and relations among the functions are the arcs.

REASONING ABOUT RESOURCE STATES

The procedure to reason about the resources' states is simple in principle. Resource Controller output data and state messages are examined to see if they fall within acceptable time and value intervals. The change or lack of change with respect to time is noted. The changes contribute to a table of derivatives that can be used to identify trends in the behavior of a resource or a set of related resources. This trend analysis serves in part as a fault detection device, allowing prediction of future conditions and intervention before some faults occur. Data for the trend analysis is collected and preprocessed locally by the monitor processes, to reduce network loading. Further fault analysis and response is performed by the System Fault Protection module.

EVENT TOKENS

The interactions of the system planner and Activity Managers produce a time ordered set of event tokens which represent steps in the schedule that indicate levels of progress. The event token format is a tag or label, a time interval and expectation values for rover state parameters. The event tokens are assembled into tree structures which are used by the schedule adapter to measure the progress of the sequence.

REASONING ABOUT THE SCHEDULE

Conditions derived from information provided by elements of the event token trees are used by the schedule adapter to trigger the scheduling calculus processes. Below are two simple examples. The first illustrates what may be a common condition in rover operation -- things taking longer than predicted.

> With the current node of the event token tree, we have reached the end of one of a series of traverse segments. The duration of a traverse segment was predicted to be 1 hour. The actual time was 1.5 hours, exceeding the specified interval. The sequence calls for an obligatory downlink in an hour. The scheduling calculus procedures are invoked to determine how the sequence and constraint envelopes for the planning of the next traverse segment are to be changed for acceptable productivity while ensuring the vehicle stops for the downlink telemetry at the proper time.

The second example is one that has the vehicle supporting a science experiment. Again, this example is an illustration of the non-deterministic nature of planetary vehicle operations. This example illustrates the requirement for the scheduling calculus to reason about the rover environment.

Objective: Take a multispectral image of a designated feature along a traverse.

Parameters: Exposure time vs. amount of ambient light, location coordinates, frequencies, objective, and sun angle.

When the rover arrives at the feature, based upon the sun angle and the position of the scan platform, the scheduling calculus procedure will reason whether there will be sufficient light during the time interval for the observation. If there is not, the method can calculate a new exposure time. However the method must also reason whether this exposure time can be used and still meet the overall objectives of the schedule.

STATUS

The scheduling calculus language currently supports the relational, interval, and functional arithmetic, as well as built-in typed functions and the ability to create typed symbols. It can perform operations illustrated by the Simmons example and will soon operate on preconstructed graphs. We are in the process of designing the database specification and interface, and the procedures for automatically constructing the graphs.

CONCLUSIONS

This rover system control architecture proposal is complex because we have tried to offer something which could eventually grow into a comprehensive solution to the problem of maximizing mission return of a rover very remote from Earth, by moving difficult, complex, time-consuming ground processes on-board. Whatever the ultimate solution to the apparent paradox among system, subsystem, and ground control, rover mission complexity will be reflected in the command and control system. It still remains to be proved (1) whether these functions can be implemented, and (2) whether the resultant implementation can be tested well enough so mission managers will allow the system to be used. We are optimistic on both counts.

ACKNOWLEDGEMENTS

The work described in this paper was carried out at the Jet Propulsion Laboratory / California Institute of Technology under a contract with the National Aeronautics and Space Administration.

While the authors have sole responsibility for the contents of this paper, we wish to extend special thanks to the following JPL personnel for ideas and review of the report: Jim Burke, Carol Collins, Sven Grenander, Ken Lambert, Jacob Matijevic, William McLaughlin, Donna Pivirotto, Dave Smith, and P. Richard Turner.

A complete list of credits would be impossible, however the authors wish to thank the following people for ideas: Charles Budney, Curt Eggemeyer, Matt Golombek, Mike Hollander, and Rob Manning.

REFERENCES

Bobrow, D., Editor, (1985). *Qualitative Reasoning About Physical Systems.*

Bourke, R., Kwok, J., and Friedlander, A. (Nov 1989). Design of a Mars Rover and Sample Return Mission, *Proc. of CNES International Symposium on Space Dynamics*, Toulouse, France.

Brooks, R., and Flynn, A., (Oct 1989). Rover on a Chip, *Aerospace America.*

Dias, W., and Gershman, R., (Oct 1987). *A Day in the Life of a Mars Rover.* Jet Propulsion Laboratory Internal Report, D-5075.

Eggemeyer, W., and Cruz, J., (to appear May 1990). PLAN-IT-2: The Next Generation Planning and Scheduling Tool, *Proc. of Goddard Conference on Space Applications of AI, 1990.*

Fesq, L., and Stephan, A., (1989). On-Board Fault Management Using Modeling Techniques, *Proc. Inter-Society Energy Conversion Engineering Conference (IECEC).*

Gat, E., Firby, R., and Miller, D., (July 1989). Planning for Execution Monitoring on a Planetary Rover, *Proc. of NASA Conference on Spacecraft Operations, Autonomy, and Robotics.* Houston, TX.

IKI - Space Research Institute (USSR) (1988). *Martian Rover Motion Control Principles Preliminary Concepts*, Pr-1422.

Johnson, S., (1983). YACC: Yet Another Compiler Compiler. *UNIX Programmer's Manual, Vol. 2.* Bell Telephone Laboratories, Murray Hill, N.J.

Lambert, K., (Oct 3, 1989). A Computational System for a Mars Rover, AIAA 89-3026. *AIAA Computers in Aerospace VII.*

Linick, T., (1985). Spacecraft Commanding for Unmanned Planetary Missions: The Uplink Process, *Journal of the British Interplanetary Society*, Vol. 28 No. 10.

Pivirotto, D. and Dias, W., (1989). *United States Planetary Rover Status-1989*, Jet Propulsion Laboratory Internal Report D-6693.

Rashid, R., (1986). Threads of a new System. *UNIX Review.* Vol 4, No. 8.

Reiners, T., (Aug 1985). *Autonomous Redundancy and Maintenance Management Subsystem (ARMMS) Executive Summary.* JPL internal report D-2414.

Report of the 90-Day Study on Human Exploration of the Moon and Mars, (Nov 1989). National Aeronautics and Space Administration.

Riethle, G., (Aug 1983). *A Summary Overview of Technology Applied to the ARMMS Demonstration Project. Fault-Tolerance Techniques.* JPL internal report D-948.

Riethle, G., (Oct 1983). *A Summary Overview of Technology Applied to the ARMMS Demonstration Project. ARMMS Executive Software.* JPL internal report D-1122.

Rokey, M., and Grenander, S., (to appear Jun 1990). Planning for Space Telerobotics: The Remote Mission Specialist, *IEEE Expert.*

Simmons, R., (Aug 1986). 'Commonsense' Arithmetic Reasoning, *Proceedings of AAAI-86*, Philadelphia, PA.

Simmons, R., and Mitchell, T., (Jul 25, 1989). A Task Control Architecture for Autonomous Robots. *Proceedings of SOAR '89 Conference.*

Smith, D., and Matijevic, J., (Oct 1989). A System Architecture for a Planetary Rover. *Proc. of the NASA Conference on Space Telerobotics, January 1989.* JPL Publication 89-7.

Wilcox, B., and Gennery, D., (1987). A Mars Rover for the 1990's, *Journal of the British Interplanetary Society (JBIS)*, Vol 40, No. 10.

N90-22314

AUTOMATED PROCEDURE EXECUTION for SPACE VEHICLE AUTONOMOUS CONTROL

by Thomas A. Broten and David A. Brown

TRW
One Space Park R2-2062
Redondo Beach, CA 90278

Abstract

Increased operational autonomy and reduced operating costs have become critical design objectives in next-generation NASA and DoD space programs. Our objective is to develop a semi-automated system for intelligent spacecraft operations support. This paper presents the Spacecraft Operations and Anomaly Resolution System (SOARS) as a standardized, model-based architecture for performing High-Level Tasking, Status Monitoring and automated Procedure Execution Control for a variety of spacecraft. The particular focus here is on the Procedure Execution Control module. A hierarchical procedure network is proposed as the fundamental means for specifying and representing arbitrary operational procedures. A separate procedure interpreter controls automatic execution of the procedure, taking into account the current status of the spacecraft as maintained in an object-oriented spacecraft model.

1. Introduction

A new generation of NASA and DoD space vehicles is emerging, for which a high degree of operational autonomy is a fundamental requirement. The requirement is typically expressed as a need for on-board task management and contingency handling for extended periods of time, without direct human involvement [Sobieski, 1989; GSFC, 1988]. The sources of the requirement stem from a) life-cycle cost control through reduction of ground support crew size (eg, for Space Station platform and mission support activities), b) improved spacecraft survivability through reduced dependence on vulnerable fixed-base ground stations (eg, for early warning and communications satellites), and c) operations at extreme distances from the Earth where signal propagation time precludes tightly-coupled human control (eg, for interplanetary probes, rovers). The increased levels of space vehicle autonomy now being proposed will substantially exceed the capabilities of traditional space/ground command link operations. The next level of autonomy will be termed the "task level" here, to distinguish it from the "command level" and to signify independent execution and control of a complete operational procedure upon receipt of an external tasking order or an internal alarm message.

Raising space vehicle autonomy to the task level involves advances in a number of areas. First, independent procedure execution means more than simply executing a prestored sequence of commands. The structure of the procedure must contain interim tests of the current situation to assure that the previous step was performed correctly before the next step is begun; otherwise, equipment damage or unsafe conditions could result. The procedure must also incorporate basic contingency handling routines, to avoid leaving the space vehicle in an undesirable state in the event of premature procedure termination.

Second, there must be facilities for on-board monitoring of the current situation that are tied-in to the space vehicle's baseline telemetry and command system. Automated real-time interpretation of data is necessary when direct human supervision is absent or delayed. Recent experiments with expert systems for automatic detection/diagnosis of anomalies, such as SCARES [Hamilton, 1986], SHARP [Lawson, 1989], SFMS [Parks, 1990], and StarPlan [Siemens, 1986] are gradually expanding the capability for real-time situation assessment.

Third, true high-level tasking implies the existence of a suitable tasking language in which a task (or operational goal) can be precisely and unambiguously expressed. Additional AI planning facilities for breaking down the high-level task into a coordinated procedure containing primitive spacecraft commands and programmed status checks will be required in the long-term. Space vehicle autonomy can benefit directly by borrowing some of the new robot task planning languages and architectures [Fu, 1987; Albus, 1987].

2. SOARS Architecture

The Spacecraft Operations and Anomaly Resolution System (SOARS) is a model-based prototype for supporting autonomous spacecraft operations. As shown in Figure 1, SOARS represents an add-on capability for incorporation into any space

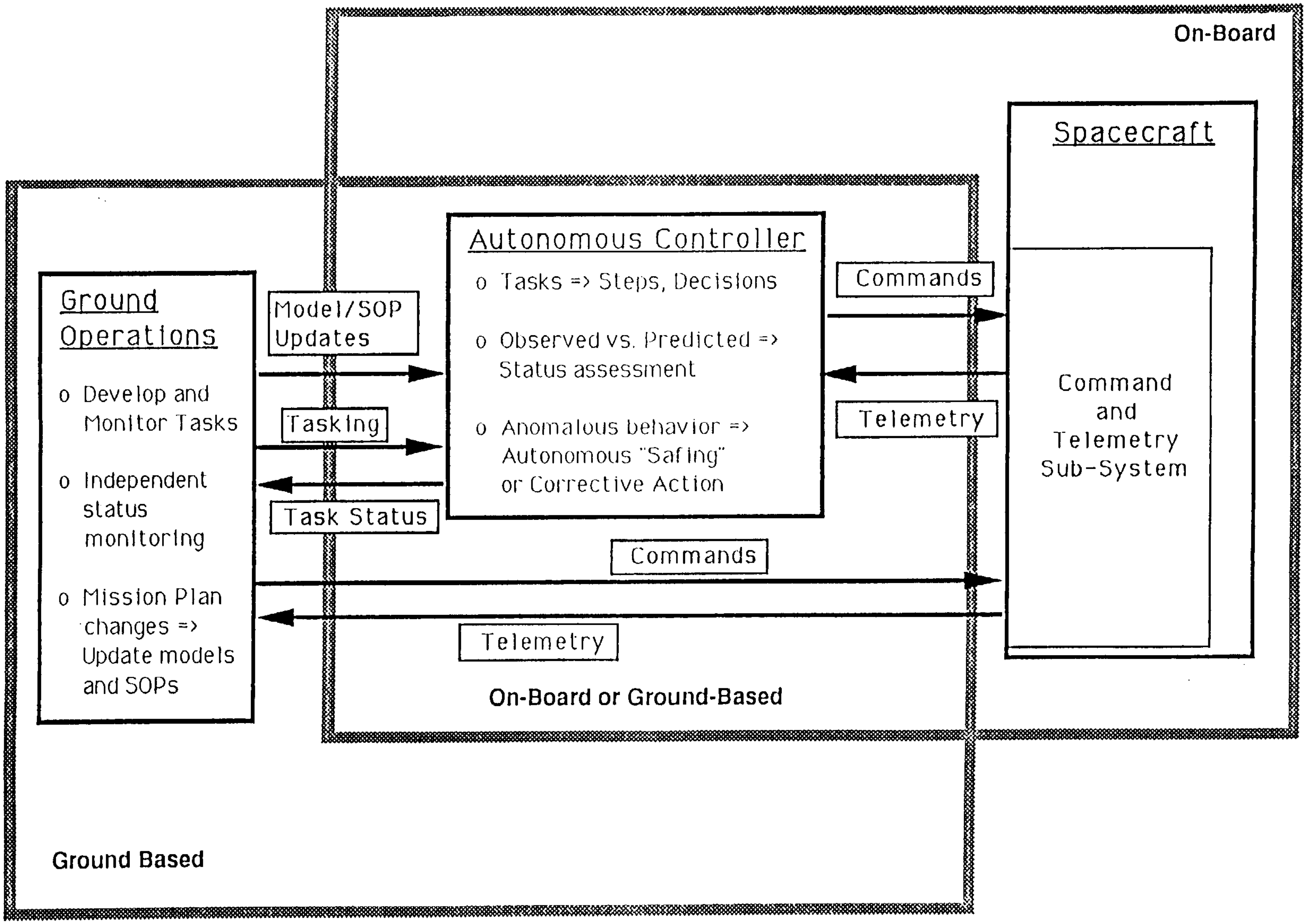


Figure 1. SOARS Architecture

vehicle/ground station control loop with standardized bus and processor interfaces to command & telemetry data. Under traditional nonautonomous operations, commands are transmitted from the ground on the uplink, and routed to the appropriate space vehicle subsystems for execution. The space vehicle subsystems continuously report status, which is encoded in telemetry for transmission to the ground on the downlink. Under upgraded autonomous operations, additional commands dealing with high-level tasking and operational procedure control can be sent to the space vehicle. Correspondingly, task status and autonomous processing performance are reported back to the ground. (Note that some additional command and telemetry slots must be allocated above the space vehicle baseline to cover autonomous operations.) For the initial demonstration flights, all of the SOARS functions would reside on the ground to permit thorough open-loop validation of the control processes. Subsequently, many of those functions would migrate to the space vehicle for true autonomous operations in the closed-loop mode.

The target SOARS flight segment, called the Autonomous Controller Unit or ACU, is composed of three basic elements, which are shown in Figure 2. The Task Analyzer receives high-level tasking commands from the ground segment (via the command subsystem), and decomposes the task into a coordinated program of primitive commands and procedural checkpoints. The Status Monitor receives status data from the space vehicle subsystems, and detects the occurrence of specified vehicle states indicating expected (ie, planned) or unexpected (ie, anomalous) conditions. The Procedure Executor executes a tasking program (received from the Task Analyzer), or a corrective action procedure (based on the detection of an anomalous condition by the Status Monitor) under real-time control.

The remainder of this paper focuses on the design and operation of the Procedure Executor component. Details of the preliminary design of the other components are contained in the ACU specifications [Brown, 1989]. The Procedure Executor interfaces with a simulated ground segment workstation called the Ground Tasking Interface, through which the operator can directly program and supervise autonomous operations. The prototype implementation runs on a Symbolics 3645 with the human/computer interface and spacecraft model (the DSP satellite) implemented in the Knowledge Engineering Environment (KEE) from Intellicorp and the Procedure Executor implemented in Common Lisp.

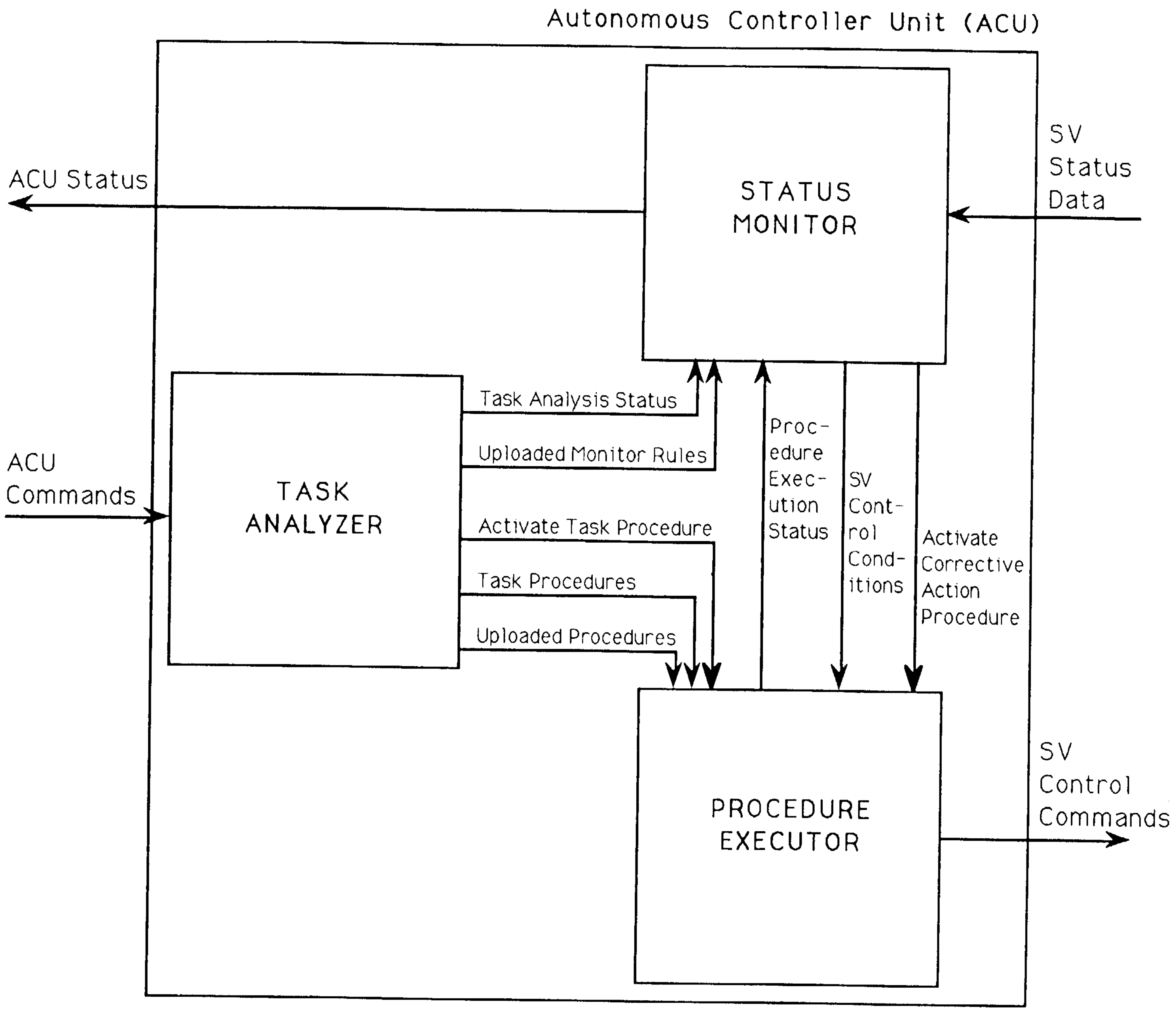


Figure 2. SOARS Flight Segment

3. Procedure Development

Figure 3 shows a sample operational procedure from the Defense Support Program (DSP). The procedure specifies how to attempt recovery from the loss of the second communications link. (For the DSP satellite there are three links total, each with a different recovery procedure. Manual procedures such as this are normally developed during detailed design and delivered as standard operating procedures [Stager, 1987].)

The procedure is entered at the top and execution follows a path through the nodes until an exit point is reached. The procedural steps specify tests to be performed (diamonds), actions to be taken (boxes), or exit points (circles). The arrows specify possible test results or simple continuations following an action. The tests and actions in some cases are primitive (eg, 'Is the L2 carrier present?' or 'Command lo bit rate switch to mode 50'); and in other cases complex (eg, 'Verify L3' -- which may involve running the complete procedure for the third link).

In the demonstration scenario a corresponding procedure graph is created on the ground using the interactive facilities in the Ground Tasking Interface (GTI). A portion of the Link-2 procedure graphic is shown in the large window of the GTI display in Figure 4. Once created by the operator, the procedure graphic is automatically processed and translated into an efficient internal representation suitable for automatic execution.

4. Procedure Representation

There are various possible ways to internally represent a space vehicle operational procedure for computational purposes. One straightforward representation is as a directly executable computer program. In this approach the program will consist of a conditional branch statement (**if** <condition> **then** <step i> **else** <step j>) or a case statement (**case** <condition> A: <step i> B: <step j> C: <step k> ....) for each procedural step. In other words, choose the next step depending on the outcome of the conditional test, until an exit point is reached.

This simple approach, however, has significant drawbacks. First, highly nested **if ... then ... else** or **case** statements are messy and notoriously difficult to understand and debug. For the sample procedure above, the longest path would require nesting to 16 levels. Second, there is no straightforward way to handle repeated steps in the procedure. Common subpaths must be represented separately and redundantly, unless **goto** statements are included (thereby overriding the natural program

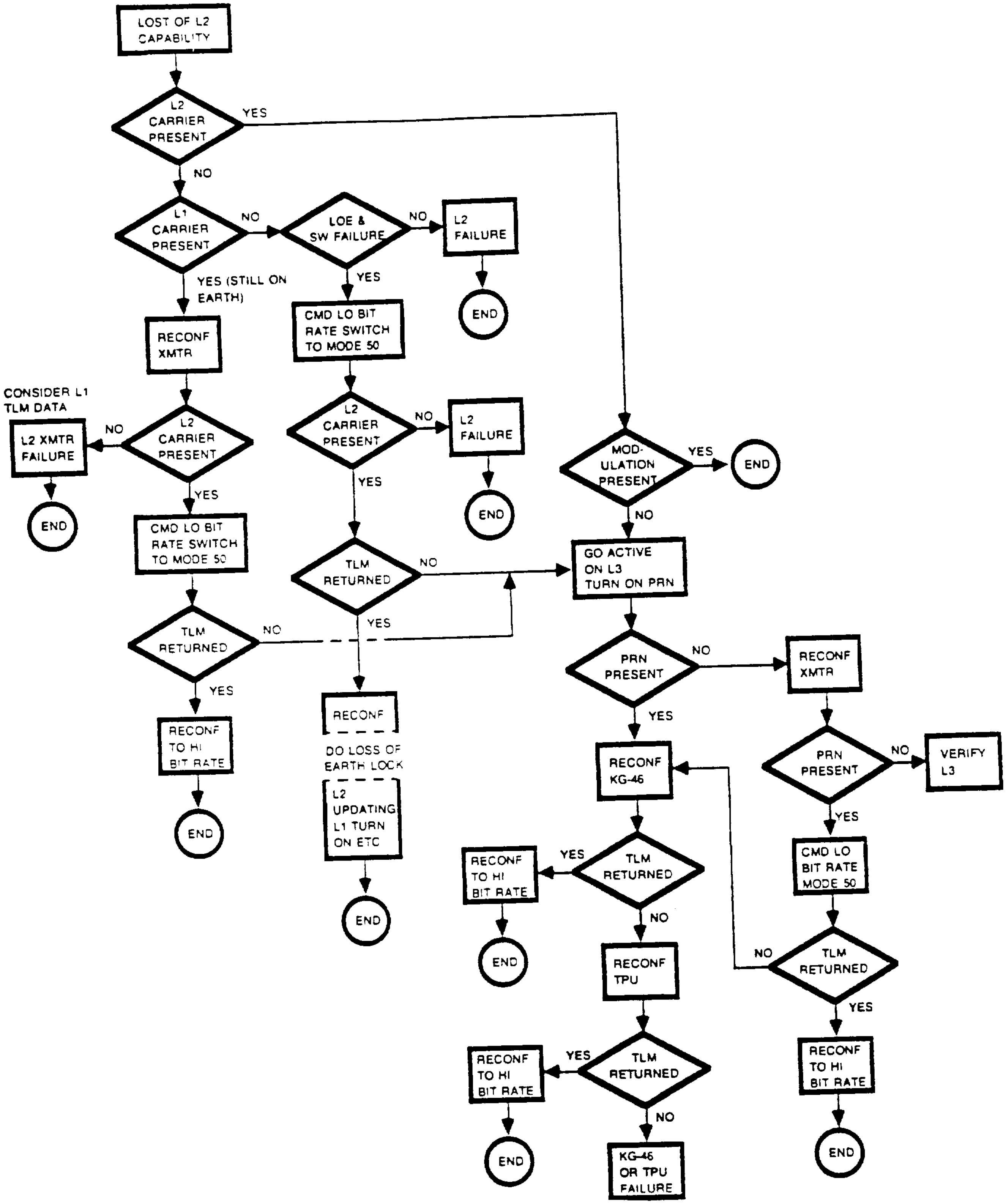
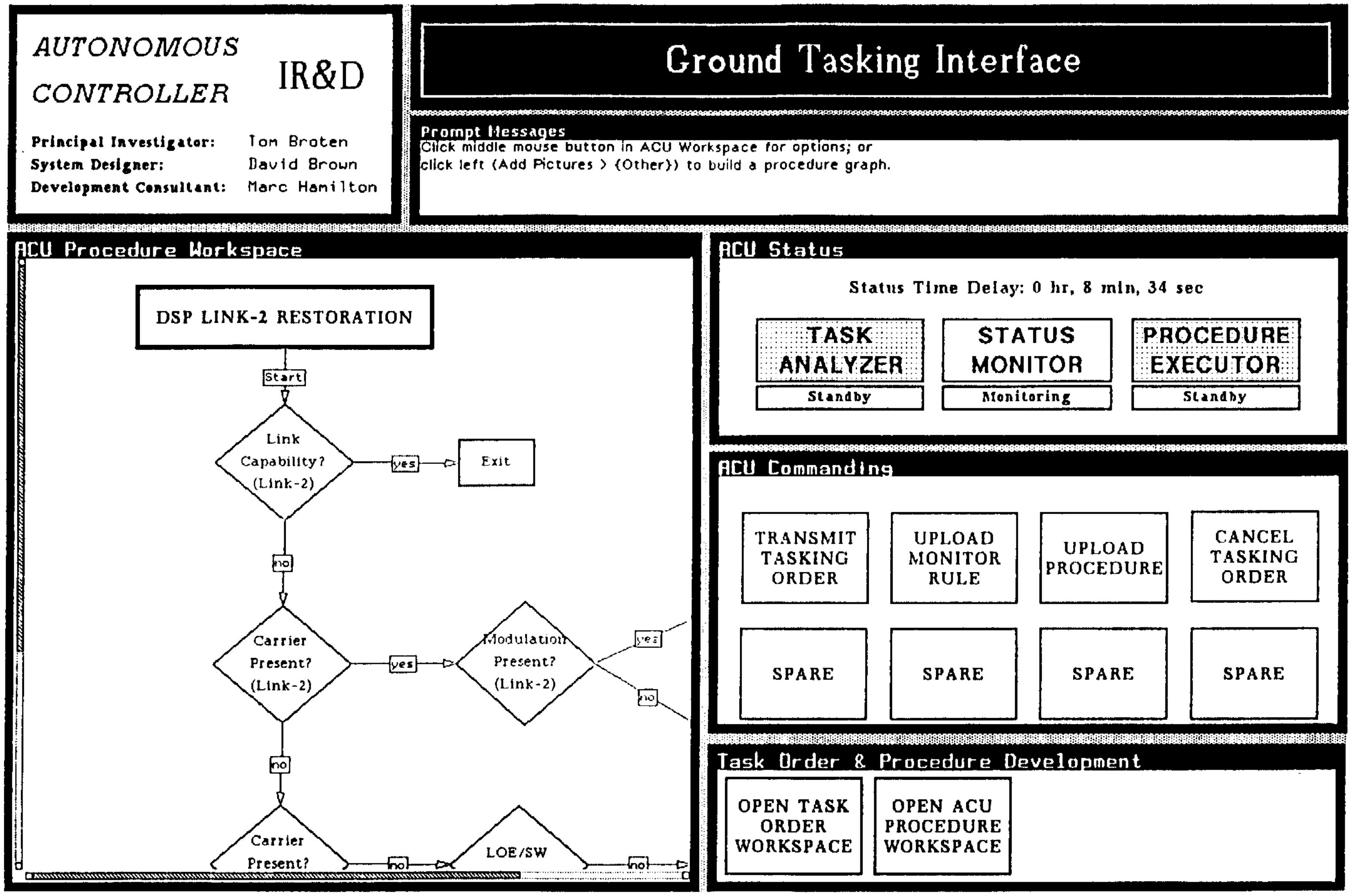


Figure 3. DSP Link-2 Diagnostic Procedure



AUTONOMOUS CONTROLLER IR&D

Principal Investigator: Tom Broten
System Designer: David Brown
Development Consultant: Marc Hamilton

Ground Tasking Interface

Prompt Messages
Click middle mouse button in ACU Workspace for options; or
click left (Add Pictures > (Other)) to build a procedure graph.

Status Time Delay: 0 hr, 8 min, 34 sec

Figure 4. SOARS Operator Interface

flow). Repetitive procedural cycles are also a problem for the same reason, with the added drawback that the depth of recursive calls is not predictable before the procedure is run.

Another possible representation for a procedure is as a collection of rules. In this approach each procedural step would be encoded in a separate **if ... then ...** rule. The rule-based representation permits a high degree of modularization of the procedural contents that is easy to understand and maintain. Which rule fires next is decided by the current situation at any time, thus no explicit procedure actually exists.

The problem with this representation is that the rules would need to implicitly encode control knowledge as well as diagnostic knowledge in order to maintain the required flow of the procedure. Mixing the two kinds of knowledge always diminishes the clarity of a rule, and defeats the original objective of rule-based systems, which is to keep these knowledge types separate. Also, not having an explicit representation of a procedure makes it difficult to reason about the procedure itself.

In keeping with the DSP example, a procedure also can be formally represented as a directed (possibly cyclic) graph. This representation appears to overcome most of the problems with the other representations while retaining their advantages, at the expense of some additional software for an interpreter (see below).

The nodes in the graph correspond to tests of various status conditions derived from the continuously updated space vehicle database model, and the arcs correspond to procedural control decisions which are followed depending on the outcome of performing a node test. Space vehicle control actions (ie, commands) are handled as side effects associated with some nodes. The actual execution of a procedure is characterized by a sequence of procedural steps (ie, a path) through the graph that provides a diagnostic record of the specific tests performed, decisions made, and actions taken. The individual nodes in a procedure graph may themselves be procedures. A hierarchical procedural representation allows complex procedures to be built up from more primitive tests and actions. For example, a Mars lander suddenly switching to an alternate landing site during the descent phase may involve resetting a number of different control subsystems, all of which possess their own reset procedures.

Moreover, the formal graph representation permits a natural separation of control knowledge (ie, what to do next) from test and actuation knowledge (ie, how to do it). The control knowledge can be stored in the procedure representation where it belongs, while the test and actuation knowledge can be placed directly in the space vehicle model which the procedure is testing and actuating.

The above definition of a procedure as a directed graph generally allows for the representation of arbitrarily complex procedures; eg, it includes procedures with provisions for contingency handling, shared substeps, repetitive cycles, interim status reporting, and even delayed operator enabling of critical steps, if necessary. It also enables high-level (AI search) analysis of the declarative procedural form using graph traversal techniques. High-level procedural analysis methods are required by the SOARS Task Analyzer component, which is responsible for selecting, adapting and generating procedures to fulfill ground tasking orders.

5. Execution Control

A procedure graph in SOARS is implemented as a hash table. The hash table allows direct access to any node given that node's identifier name. The content of a node, in turn, stores the name of a primitive function which is to be executed when that node is activated (eg, a command or conditional test), and which node to branch to next given the real-time result of the command or test. A hash table is automatically created for a procedure after the operator develops the graphical representation of the procedure using KEE's interactive interface facilities .

The execution of a procedure graph (whether for verification purposes on the ground or for operational purposes on the spacecraft) is controlled by an interpreter module. The procedure interpreter takes as input the hash table representation of a procedure. It begins execution at the designated starting node and follows a path through the procedure graph. To execute a node, the interpreter looks up the name of the primitive function associated with the node in a telemetry/command function library. The library contains the primitive functions for performing all of the basic telemetry tests and command sequences for the spacecraft. The library, which will be different for each spacecraft, serves as the principal interface between SOARS and the spacecraft via the provided command & telemetry subsystem. The library functions are written in the Tell and Ask (tm) pseudo-English language provided with KEE to simplify the interface with domain experts. After executing the appropriate function for the node and receiving the returned results, the interpreter steps to the next node based on those results.

Procedure execution stops when a termination node is reached.

Throughout the execution of a procedure and at critical designated checkpoints, the Procedure Executor reports status back to the Ground Tasking Interface and to the on-board Status Monitor. The operator and Status Monitor together exercise external (high-level) supervisory control over the execution process.

6. Directions for Future Work

Our ultimate goal is to field an integrated system for intelligent spacecraft operations support, including a standardized SOARS shell and integrated Knowledge Capture Tools (KCT). The KCT developments of the prior year are targeted to support spacecraft design, integration & test, and operations through the coordinated collection and refinement of analysis and simulation models. The KCT comprises the off-line knowledge-base design component, while the SOARS shell represents the on-line operational component of the integrated system.

During the coming year we intend to produce a working prototype of KCT/SOARS and release it to one or more project teams for use in creating spacecraft models and evaluating simulated autonomous control. We plan to develop demonstrations that will focus on the Advanced Tracking and Data Relay Satellite (ATDRS) and a Polar Orbiting Platform (POP). The procedural control and high-level tasking elements of this year's development will be combined with the telemetry monitoring elements of the Spacecraft Fault Management System, or SFMS, (a currently operational real-time expert system) to complete the prototype SOARS architecture. The current SFMS, which is based on symptomatic analysis of fault conditions, is being upgraded to handle model-based fault diagnosis.

Once developed, KCT/SOARS will provide spacecraft designers and operators with an integrated prototyping environment for capturing and then using spacecraft data and models. The initial prototyping environment is specifically aimed at a) generating simple, well-structured models which can be later expanded; b) diagnosing a small number of faults to a level where a certified automatic procedure can reliably rectify or evaluate the condition; and c) providing a straightforward capability for high-level tasking (ie, simple on-board procedure activation) from the ground. Fully automated corrective action procedures, in general, will not be provided; instead, automated control will be limited to the execution of simple diagnostic procedures capable of providing backup information. We believe a careful, gradual approach to increased levels of satellite autonomy is the only viable approach, considering the potential consequences of precipitous actions.

Cited Works

Albus, J.S., McCain, H.G., Lumia, R.L., NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM), National Bureau of Standards, 13 Mar 87.

Brown, D.A., Software Requirements Specification for a Space Vehicle Autonomous Controller, TRW Memo, 23 May 89.

Brown, D.A., Autonomous Controller Block Diagrams, TRW Memo, 31 May 89.

Fu, K.S., Gonzalez, R.C., Lee, C.S.G., Robotics: Control, Sensing, Vision, and Intelligence, McGraw-Hill, New York, 1987.

Goddard Space Flight Center (GSFC), Flight Telerobotics Servicer (FTS) Requirements Document, 1988.

Hamilton, M., Dignam, F., Murrin, J., "SCARES: A Spacecraft Control Anomaly Resolution Expert System", Proceedings of Expert Systems in Government Symposium, IEEE, Oct 86.

Lawson, D.L., James, M.L., "SHARP: A Multi-mission AI System for Spacecraft Telemetry Monitoring and Diagnosis", Goddard Conference on Space Applications of Artificial Intelligence, May 1989.

Parks, K.G., Broten, T.A., Rathe, R.A., "SFMS: An Expert System for Spacecraft Operations", Second Annual Conference on Innovative Applications of Artificial Intelligence (IAAI-90), May 1990.

Siemens, R.W., Golden, M., Ferguson, J.C., "StarPlan II: Evolution of an Expert System", Proceedings Fifth National Conference on Artificial Intelligence (AAAI-86), Aug 86.

Sobieski, S., Customer Data and Operations System (CDOS) Operations Concept Document Version 1.0, Goddard Space Flight Center, Jun 1989.

Stager, D.C., Satellite Contingency Analyses and Schematics: Satellites 0014 - 0017, TRW Report # 45021-321-710A3-018, 2 Mar 87.

Special Topics

Using Expert Systems to Implement a Semantic Data Model of a Large Mass Storage System

Larry H. Roelofs
National Space Science Data Center
Computer Technology Associates
Rockville, Maryland 20852

William J. Campbell
National Space Science Data Center
National Aeronautics and Space Administration
Goddard Space Flight Center
Greenbelt, Maryland 20771

Abstract

The successful development of large volume data storage systems will depend not only on the ability of the designers to store data, but on the ability to manage such data once it is in the system. Our hypothesis is that mass storage data management can only be implemented successfully based on highly 'intelligent' meta data management services. Such services would allow database administrators and users to manipulate, update, and access data, and related information and knowledge in a logical manner and yet is powerful enough to support the performance needs of a large mass store system. Historically, there have been attempts at building data management services for very large volume data systems, however, when the amount of data being managed got large the meta database itself failed as a consequence of its own size and complexity.

There now exists a proposed mass store system standard proposed by the IEEE, that addresses many of the issues related to the storage of large volumes of data, however, the model does not consider a major technical issue, namely the high level management of stored data. However, if the model were expanded to include the semantics and pragmatics of the data domain using a Semantic Data Model (SDM) concept the result would be data that is expressiveness of the Intelligent Information Fusion (IIF) concept, the result would be data that is organized and classified in context to its use and purpose. The implementation of a SDM requires the application of AI and related computer science technologies such as object oriented representation, property inheritance and rule based decision making. Presently there does not exist unique software for developing SDM's that address the complex representation of data meta data and related information and knowledge.

This paper presents the results of a demonstration prototype SDM implemented using the expert system development tool NEXPERT OBJECT. In the prototype, a simple instance of a SDM was created to support a hypothetical application for the Earth Observing System, Data Information System (EOSDIS). The massive amounts of data that EOSDIS will manage requires the definition and design of a powerful information management system in order to support even the most basic needs of the project. The application domain is characterized by a semantic like network that represents the data content and the relationships between the data based on user views and more generalized domain architectural view of the information world. The data in the domain are represented by objects that define classes, types and instances of the data. In addition, data properties are selectively inherited between parent and daughter relationships in the domain. Based on the SDM a simple information system design is developed from the low level data storage media, through record management and meta data management to the user interface.

Background

In the past decade, operations and research projects that support a major portion of NASA's overall mission have experienced a dramatic increase in the volume of generated data and resultant information that is unparalleled in the history of the agency. This information glut is growing nonlinearly due to the increasing number and quality (higher resolution) of sensor systems and is expected to continue to accelerate in this fashion for the foreseeable future. The effect of such large volumes of data is that without a significant improvement in information technologies there is no assured way that desired data can be managed, identified and accessed.

If nothing is done to reverse this process, large data archives will evolve that may contain very valuable data for which there is no easy way to find desired data. The reason being is that the search times to find desired files will be prohibitive as selection will require human interpretation and user queries will not map logical into the data being stored. For example, in the Earth Observing System (EOS), which is a major new NASA project supporting the "Mission to Planet Earth" program, data volumes produced could be as high as four Terabytes per day, with much of the data being spatial in nature. [1] If such volumes are accumulated over the life of the mission (planned for 15 years) the total data volume may be as high as 30 Petabytes (30,000 trillion bytes) when backup and data reprocessing are included.

In addition to the amount and kinds of data, the number of professionals in the application disciplines is not expected to increase significantly enough to resolve this data/information selection problem. Thus the dilemma arises that the amount and complexity of data and information system will exceeded the ability of scientist and engineers to understand and take advantage of them. [2] Based on the scope, expected growth and dominance of the data volume problem, it is anticipated that the future ability of NASA to say apace with the data it collects from space and Earth science programs will be significantly affected by its ability to manage and use such data in an efficient and timely fashion.

The Limitation of Information Systems for Mass Storage

Present database management systems cannot support the low level performance needs of a mass store system, and at the same time provide the capability to the user to identify, select and access desired data quickly and easily. The result is that the design of an information system for a large mass storage system must be separated both operationally and functionally into two elements: (1) Low Level Record Management and (2) High Level Meta Data Management and User Interface (Figure 1). The low level record management component must be biased toward performance and must focus on storing and managing file storage records at the device or hardware level. The high level meta data management system must interface to both the low level record management and user interface components and provide a robust data management and access to the files in the mass store system. In addition, such a system should allows the user to access the data without understanding the data domain architecture, data content or data query language.

The rationale for separating the low level record management from the rest of the database system is that the mass store system performance and the large number of individual records limit the kind of

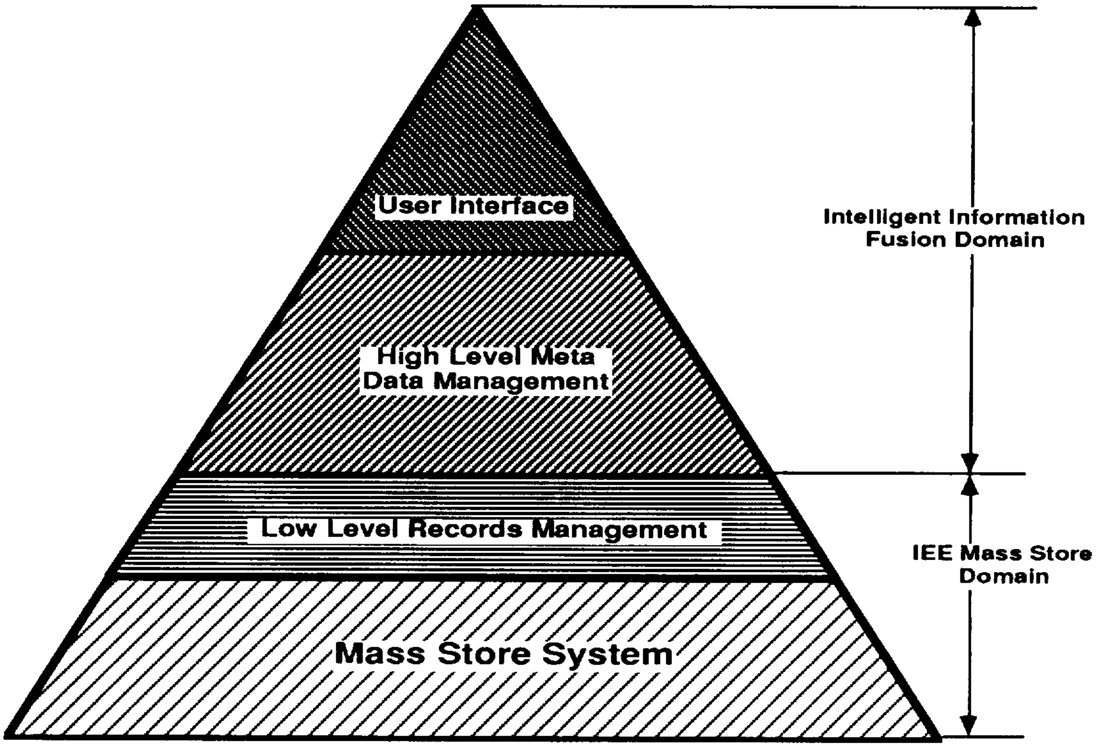

User Interface
High Level Meta Data Management
Low Level Records Management
Mass Store System
Intelligent Information Fusion Domain
IEE Mass Store Domain

Figure 1
Mass Store Information Management Functional Architecture

data management system that can be implemented. Such a system must be simple in order match the access and I/O performance of the storage hardware and at the same time be easy to update and manage. Since the low level meta data system is not intended to support user access, it will be necessary to implement a separate high level meta data system that can interact with the low level meta data system and at the same time interact with and support the user interface. Using present database technologies the system would most likely use a relational database system, even though it would impose limitations on the capabilities of the mass store system in terms of data access and ingest [3]. There three reasons for this selection: the technology has significantly grown in performance and capability over the past ten years, such that it now provides significant benefits over databases that are based on the other two data models (network [4] and hierarchical [5]); it is a fairly mature technology that is well understood, easy to implement and easy to maintain; there are a large number of commercially available software systems that run on most of the popular computers available today.

However, once the decision is made to implement the high level meta data management system using a relational database there are significant representation and data manipulation issues inherent in relational technology, namely:

1. Database architecture based on a flat file structure
2. Database abstraction limited to aggregation
3. No way to capture and manage meta knowledge (procedural and control information)
4. No way to support the representation of time

5. No way to deal with class-type and derived data where it is required to support the generalization of the class
6. Difficult query language based on relational calculus that is not easy to use or understand
7. The database requires normalization in order to function efficiently.

The first issue can be explained by considering that in a relational database management model, the tuples in a relation correspond to the records in a file and the tables are simply flat lists that can be manipulated only by table joining. [6,7] The second, third, fourth and fifth issues are a consequence of the mathematics that the database model is based on. Essentially, the only structure that a relational database supports is aggregation (one to one, one to many and many to many), not class, type or time. [8] The sixth issue is also a consequence of relational calculus which requires very precise syntax and structure in order to form a mathematically correct query. Such a query language usually requires a rather sophisticated user to form even the most simple requests. [3]

The last issue is the result of the need to optimize the database design in order to maximum performance and minimize maintenance. [9] However, normalization usually results in most of the semantics and pragmatics of the data domain being removed. The removal process is the consequence of redundant, but logically related, attributes being deleted from common relations (tables), such that an attribute exists only once in the database. This removal process logically fragments the database in a manner that makes the database difficult to understand, to the casual user, without a design map (e.g. E/R diagram). If this is combined with the lack of expressiveness of the Database Manipulation Language (DML), it is no wonder that few large database systems are understandable by the users' except through some simplistic interface that more often than not severely limits the users' access to the data.

Given the above, the only recourse is to put all the semantics, pragmatics, procedural and operational knowledge into a user interface which must be totally customized to accommodate each application or user view. Since the interface is usually implemented after the database has been designed and populated, its architecture tends not to be related to the database architecture in any way.

A Proposed Solution

Considering the data volume problem, it would appear that large mass store systems will be needed to store the flood of data that will be collected over the next fifteen to twenty years. However, traditional database technologies will not be able to support the high level meta data management needs of such a system, and at the same time have the performance capability necessary to handle low level file management. Consequently, alternative information management strategies that implement the management and access of data, meta data and supporting information and knowledge in a coherently structured manner must be employed. Such a structured approach to data and information management we call Intelligent Information Fusion (IIF). The IIF concept is based on a layered architecture that supports the semantics, pragmatics and syntax of the data domain from the system, data and user points of view so that data can be input, managed and accessed in the most efficient and appropriate manner. A diagram representing the overall IIF concept, in the context of an EOS archive, is shown in Figure 2.

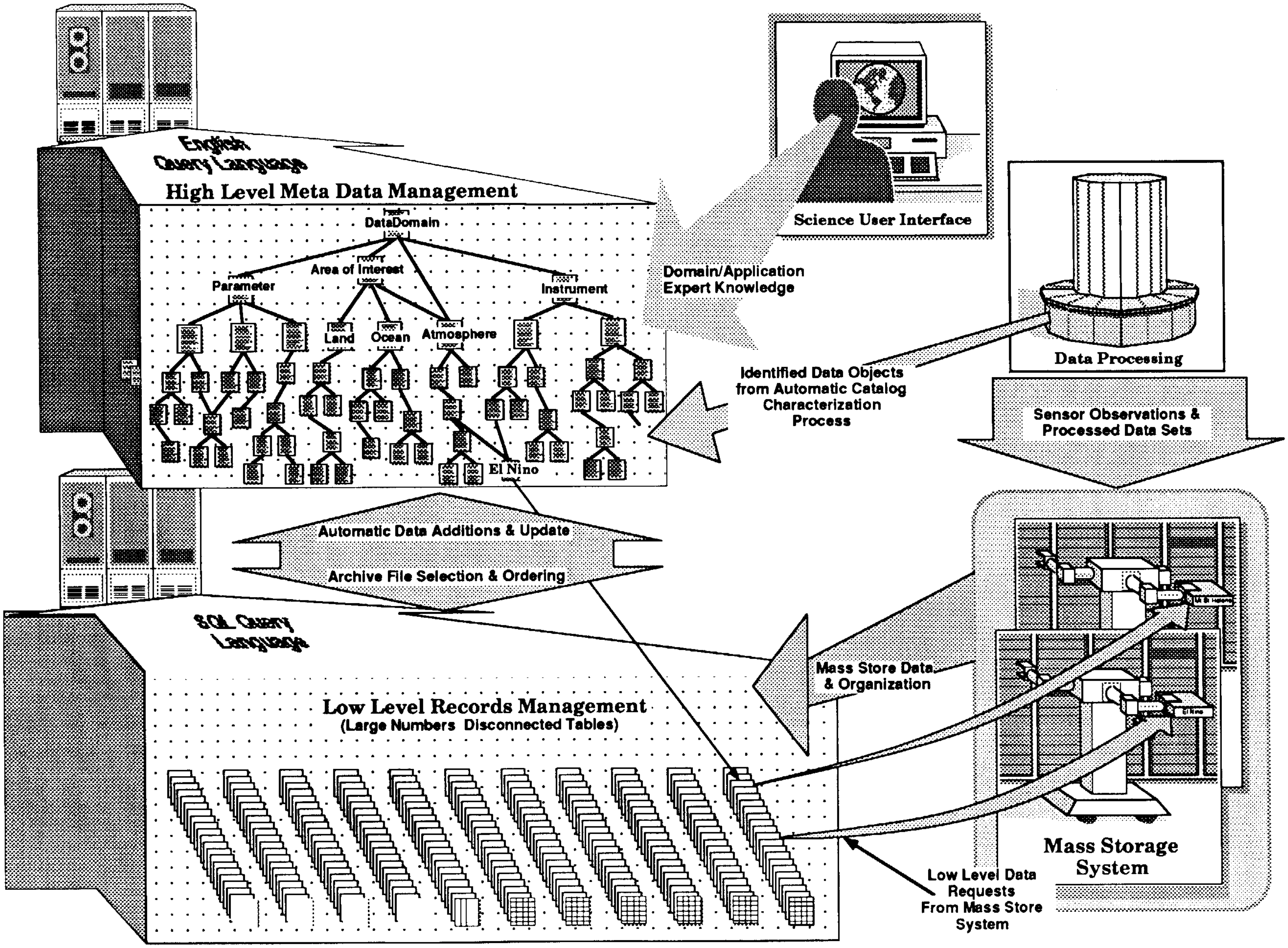

English Query Language
High Level Meta Data Management
DataDomain
Parameter
Area of Interest
Instrument
Land
Ocean
Atmosphere
El Nino
Science User Interface
Domain/Application Expert Knowledge
Data Processing
Identified Data Objects from Automatic Catalog Characterization Process
Sensor Observations & Processed Data Sets
Automatic Data Additions & Update
Archive File Selection & Ordering
SQL Query Language
Low Level Records Management
(Large Numbers Disconnected Tables)
Mass Store Data & Organization
Mass Storage System
Low Level Data Requests From Mass Store System
El Nino

Figure 2
IIF Concept With An EOSDIS Archive

The IIF concept consists of three essential elements that are absolutely necessary to the successful implementation and operation of a large mass store system, these are:

1. *Semantic and Knowledge Based Representation*, which we call information synthesis is the process that captures the essence of the data enterprise (i.e. database domain) at all three levels of data representation (semantic, pragmatic, context, and syntax) for the generalized data domain. It addresses functional and operational views, from the highest level class structure, to intermediate meta data, to the lowest level data granule or file.

2. *View and Application Representation* supports multiple discipline, user, and application goals, based on the understanding and needs of the users.

3. *Automated Data Cataloging and Characterization* supports the addition and updating of data to the mass storage system while automatically updating the supporting meta data to reflect changes and additional data. [10]

The first element is the most important for the implementation of a large mass store system since it provides the organizational structure of the storage system, and its supporting record management system. In addition, it also provides input and direction to the design of the higher level meta data and user interfacing that supports the data access and ingest operations.

Information Synthesis and Semantic Data Models

Information synthesis, the context of the massive data storage problem, means the structuring and fusion of data, information, knowledge, and meta knowledge into a coherent structure that can logically be used for a particular purpose. The information synthesis concept is based on the semantic data model, enhanced with a knowledge base that can provide the control and operational strategies needed to support a complex information domain. The semantic data model was proposed in the late seventies by McLeod and King, [11] and refined by Potter, and Trueblood. [12] The model draws on research from the fields of Database Management (DBM) and Artificial Intelligence (AI) in performing data domain or enterprise modeling. The model is intended to address and overcome the problems of domain representation limitations that exist in the three traditional database models [13]; the hierarchical , network, and relational. All three of these models are basically record oriented with very complex data manipulation languages. [9] Their development was motivated and influenced by primitive file system implementation concerns that limit the size and volume of such systems. [14]

The development of models that provide more user oriented modeling flexibility, without being constrained by an implementation structure, was the primary goal of early semantic data model researchers [15, 16]. The resulting models provided a "natural" mechanism (e.g. similar to the way a user or system designer views an application) for specifying the design of a database which more accurately represents the data and relationships among the data than the traditional models. [12]

The IIF concept allows database designers to represent the objects of interest in the proper context and their relationships in a manner that more closely resembles the view that the user has of these objects and relationships. [8] Consequently, designers are freed from the necessity of having to model a variety of different relationships with only one modeling construct, which typically results in the loss of their exact meaning. The end result of using limited constructs is that the application design falls short of modeling the user's actual situation or needs.

One of the most important features of the IIF concept is that it provides powerful abstraction constructs, such as generalization and aggregation, that are not found in the traditional data models. [17] Generalization allows the designer to group similar objects together in order to concentrate on the more general group object. Aggregation allows the designer to model an abstraction object based on the properties or attributes on the object. [8] Other constructs that deal with time and space have also been associated with semantic data modeling. [16]

In addition to generalization and aggregation, semantic data models are characterized by the notion of "derived or virtual data." [18, 19] Essentially, derived data can be thought of as data values which are not actually stored in the database but are produced or derived when needed from existing data and relationships. This concept is very important for mass store systems since it can potentially reduce the stored data volume significantly, because the storing of explicit values is unnecessary in as much as the relationships and means exist for deriving this data.

Semantic data models may be classified according to one of three general categories: relational, functional and semantic network. Models in the relational category are basically extensions to the relational database model. The functional models have an equally strong mathematical influence. In the functional model, the function is the primary notion used to represent and manipulate objects and relationships. The third model category is the semantic network model which has a close relationship to the semantic network knowledge representation formalism found in the AI field. However, unlike the early network models it was necessary to enhance the model with meta knowledge about the domain that describes how the data is manipulated and acted on, as proposed by Potter and Kerschberg. [12,20]

An Intelligent Information and Fusion Prototype

To prove the usefulness of IIF concept to the mass storage information system design, a demonstration prototype effort was undertaken using existing technologies. A cursory survey of the market place was conducted to identify any existing semantic database system tools. The result was that there are few commercially available systems, none which provide the robustness necessary for supporting large mass storage operations. Therefore, the only alternatives were either to develop new customized modeling tools or to use existing related technologies to implement such a tool. The second alternative was selected because previous experience has shown that there is a higher probability of success if several existing technologies can be combined to create a new single integrated tool.

An initial prototype system was conceived, based on a Macintosh II computer, using the expert system development tool, NEXPERT, built by Neuron Data Inc. NEXPERT provided many features that were necessary for semantic data modeling including:

- Frame based representation for creating data objects, data classes super classes that represent the content of the data domain
- Inference engine that supports rule strategies that employ pattern matching and forward and backward chaining for capturing the meta knowledge and procedures that support specific operational agendas
- Object representation that supports, properties, meta slots and multiple inheritance in both directions for supporting the passing of data properties and values between data objects and classes.
- Graphical interfacing for displaying, in a easy to understand manner, the complex network structure that represents the data domain. [21]

In addition to the development tool, it was also necessary to find or create a hypothetical database for modeling. The second option was selected since this was only a demonstration prototype effort. However, in order to understand how scientific databases are designed, two operational databases were reviewed, the Pilot Land Data System (PLDS) and the NASA Climate Data System (NCDS). Both systems store their sensor data off line and use the ORACLE DBMS for management of the meta data which is structured into two levels of abstraction. Namely, a Data Catalog that provides a general over view of stored data sets and a Data Inventory which provides detailed information about each stored data set.. In addition to the Data Catalog and Data Inventory meta database structure, both have an interface based on the Transactional Application Executive (TAE) software, with the NCDS enhanced to include some data visualization.

Besides understanding how existing data systems manage their data and meta data, a atmospheric scientist was interviewed to determine how she perceived the overall EOS data domain, as well as her specific areas of interest within the domain. The result of this effort was the definition of a simple structure for studying cloud cover, as well as the identification of three critical design goals that the model should be able to support. These goals were:

1. Where are all occurrences of a specific data object (e.g. episodic event, El Nino)
2. What data objects have been found for a specific observation.
3. What data objects have been found to occur, or change, over time for a specific location.

Semantic Model Design

Based on a review of the two data systems studied, the atmospheric scientist interview and a review of the EOS program, a hypothetical EOS data domain was formulated and is presented in Figure 3, Semantic Model Top Level View. The domain consisted of the three major data areas (Instrument, Interest Area, and General Parameter) that could be used to organize and find data as well as a more detailed structure to support a specific area of interest (e.g. atmosphere and cloud cover). The specific area of interest selected for the prototype was atmosphere, with the sub area clouds.

In addition to an overall data domain structure there were six key design considerations identified that needed to be accommodated in the data modeling process. These considerations were:

1. All low level data elements or granules had to inherit the property slots, and where appropriate, the values associated with the slots from its parent class object.

2. All first level data objects that related to a specific sensor observation file in the low level records management system must have properties associated with the object that represent the important features observed in the record, as well as all ephemeris and supporting ancillary data.

3. Rules are required to guide the ingesting and summarizing of data set header data (name, data, time, location, etc.) from the low level data objects to the object's class frame. [22]

4. All low level data objects and their associated sensor observation file have to be accessible by all logical paths.

5. Must be able to interact with a relational database system to allow it access and read low level data in the records management layer.

6. Must be able to deal with both the generalized data domain organization as well as supporting unique applications that use the data for some specific goal or purpose.

A data model based on the data domain concepts was formulated. The model included the three major information components that make up a data domain: the operational or database view, the application or semantic view and meta knowledge and control. In addition, the operational view was driven by three design considerations: the creation of a skeleton structure upon which the rest of the model could be implemented, the representation of the database domain that it supported the management, access and ingest of the actual data objects (individual sensor observations), and a strategy to support the access of data by instrument.

Construction of the Prototype

STEP 1 - Formulation and Characterization of the Operational View

The first step in the operational view representation was to identify the major class objects in the data enterprise and determine how they were related. Once this information was determined, it was input into NEXPERT by characterizing each class object with a frame that defined its daughter classes, as well as any properties that needed to be associated with the frame. In this step the top level database object was defined as, *Enterprise*, with primary super class objects (or domains): *Instrument, Area of Interest,* and General *Parameter*. In some since these three super class objects represent more than just the high level abstraction of its daughter elements because they must also include meta and procedural knowledge. The rationale for selecting the above super classes were:

- **Instrument** - The low level data needs to be stored and managed by instrument so an overall super class object must be created that deals with the topic area of instruments.

- **Area of Interest** - The user can select data by area of interest, where the focus is on some property within the area, sub area or component (e.g. cloud type at some time and/or location). This super class object is part of the application view which is the logical area where

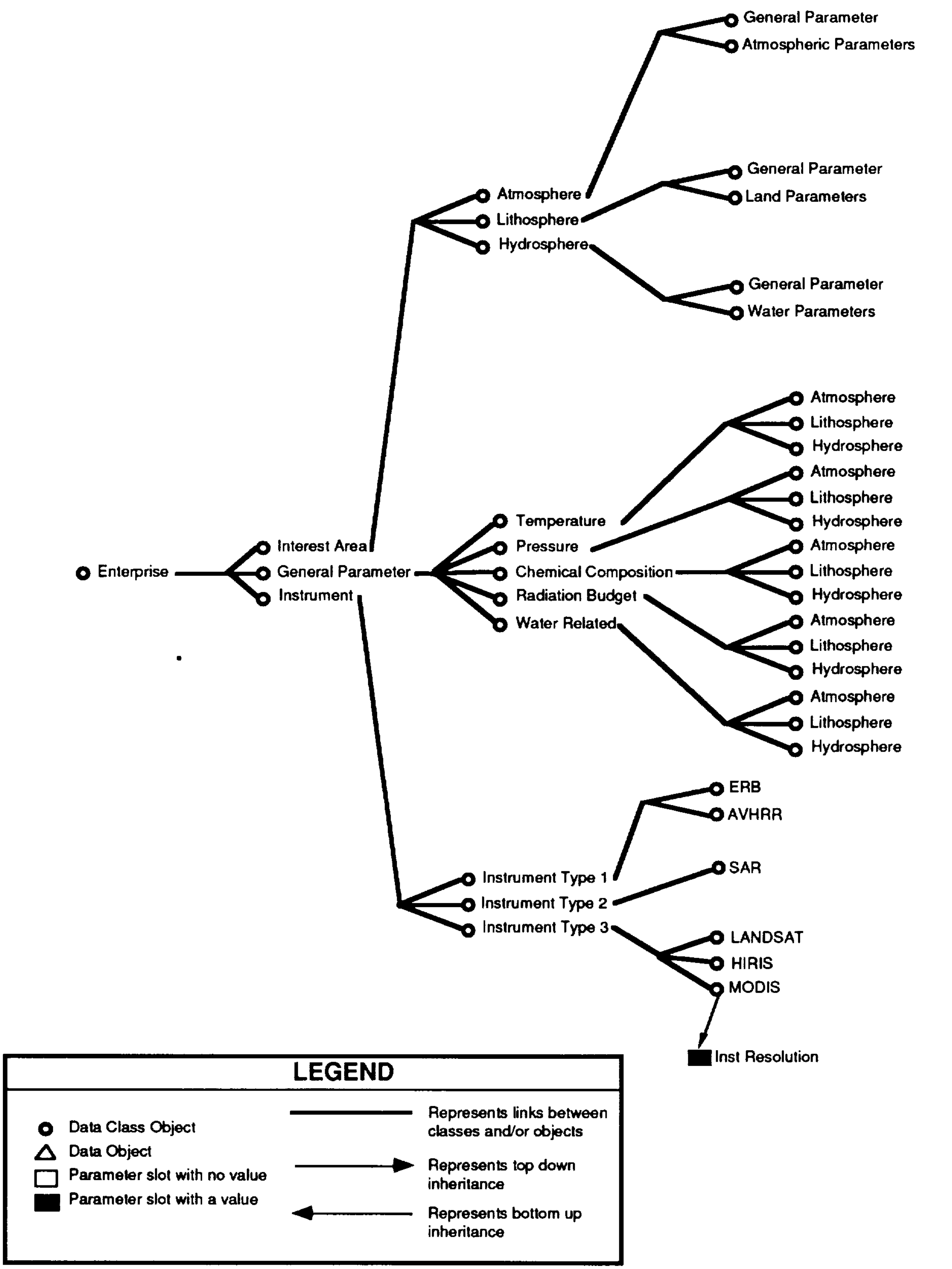


Figure 3
Semantic Model Top Level View

discipline, feature objects (data objects found with in a sensor observation) and supporting scientific information should be located.

- **Parameter** - Data can be selected by parameter across multiple disciplines or areas of interest (e.g. temperature for some location for land, atmosphere and water). This super class object is an alternative path that users require when the focus is more parameter and less single interest area oriented.

After the first super class objects were selected, each was expanded into several lower level daughter classes using generalized information gained from discussing the problem with scientists involved in atmospheric science and scientific database system design. A diagram of the result of this step is presented in Figure 3, Semantic Model Top Level View. Of particular importance is how the domain can only be represented by a network. This seemed rather obvious to a user until one tries to use the graphics tools of NEXPERT and finds that what is presented is a tree that is session dependent. However, upon closer inspection the super class object *General Parameter* was also found to be a subclass object of each of the subclasses in the *Area of Interest* super class object. Thus *General Parameter* serves not only as a high level super class object, but it is also a daughter object for each of the sphere objects (*Atmosphere, Lithosphere, Hydrosphere*) in the *Area of Interest* super class object .

Under the super class object *General Parameters,* representative physical parameters were identified and selected that appeared to cut across discipline and unique science boundaries. This was considered important because of the need to study a single parameter for more than one discipline. Although it is possible to look by *Interest Area,* by *sphere* and by *General Parameter,* this query would produce a large amount of non-relevant data since there are many more antecedents involved in defining what data or information is desired.

Under the super class object *Instruments* three classes were created that are used to partition this part of the domain into more manageable groups. This structure approximates what needs to be done in order to efficiently perform a query based on instrument type.

STEP 2 Formulation and Representation of the Instrument Super Class Object

The second step in the modeling process was the complete representation of the *instrument* super class. This is of critical importance since this is where all of the data sets and related sensor observations are located and subsequently managed. A singularly important consideration was that the model had to accommodate the automatic input of meta data and identified feature data into the database as new data objects were added to the mass store system. The impact of this requirement was profound, as it required a property inheritance strategy to function both from the top down as well as the bottom up.

NEXPERT supports a robust property inheritance capability by providing either top down, bottom up or both. Basically properties are included into each frame's property list by name. After the name is entered the system asks for the property to be defined by type as either a strings (variable length), integer, floating point, date, time, or boolean value. After the slot is defined, an inheritance strategy is selected and a property value is put into the slot. If the parameter is to be inferred, then the slot will defined as empty and the value will be determined based on some set of rules. (It should be noted that at the beginning of the model development session the breadth first strategy was selected as the default.)

Property slots were needed for the data model for three purposes. First, they were required to allow the passing of relevant class type information to the low level objects that were associated each unique data records in the mass store system (information such as sensor resolution, sensor type, etc.). Second, they were required to store summary information at the class level based on the inference of all the related parameters of the daughter objects. Third, they were required for storing 'identified features', detected and cataloged as part of the Automatic Data Cataloging and Characterization process. [10]

The property slots without values, were inferred by rules for the following cases:

- To determine meta data for a data object from the header record of the data file stored in the mass store system. Meta data consisted of ID number, date, time, observation location, etc.

- To determine summary information properties that would be required for some parent object. Information inferred included: data range, time range, number of data records, etc.

- To store identified features found in the data file (sensor observation) from the results of the Automatic Data Cataloging and Characterization process. Identified features include such things as clouds, volcanos, thunderstorms, lakes, forests, etc.

Once the *instrument* super class domain representation was complete (shown in Figure 4), sets of rules were coded to support the summarization of the inheritable properties from all the daughter data objects into property slots at the class level. For example, rules were created that read the "date value" from each data object's "date" property. These values were then stored in a list, and sorted in ascending order. When this was complete the top value would be the earliest date and the bottom the latest date. When the two values were combined they provided the date range that was then placed in the class's date range slot. Using a similar strategy all of the class's other summary property slots were populated.

STEP 3 Formulation and Representation of the Application View

The application view consists of the discipline domain which is characterized by areas of scientific interest, and science object domain which contains all classes and objects that are studied and measured as part of a scientific endeavor. There is a close logical link between any discipline domain and its related objects of interest. However, they are separate in that a discipline domain uses observations from instruments to study phenomena that are the objects of interest. The discipline observations are actually the same as instrument sensor observations with some value added processing such as calibration correction and registration.

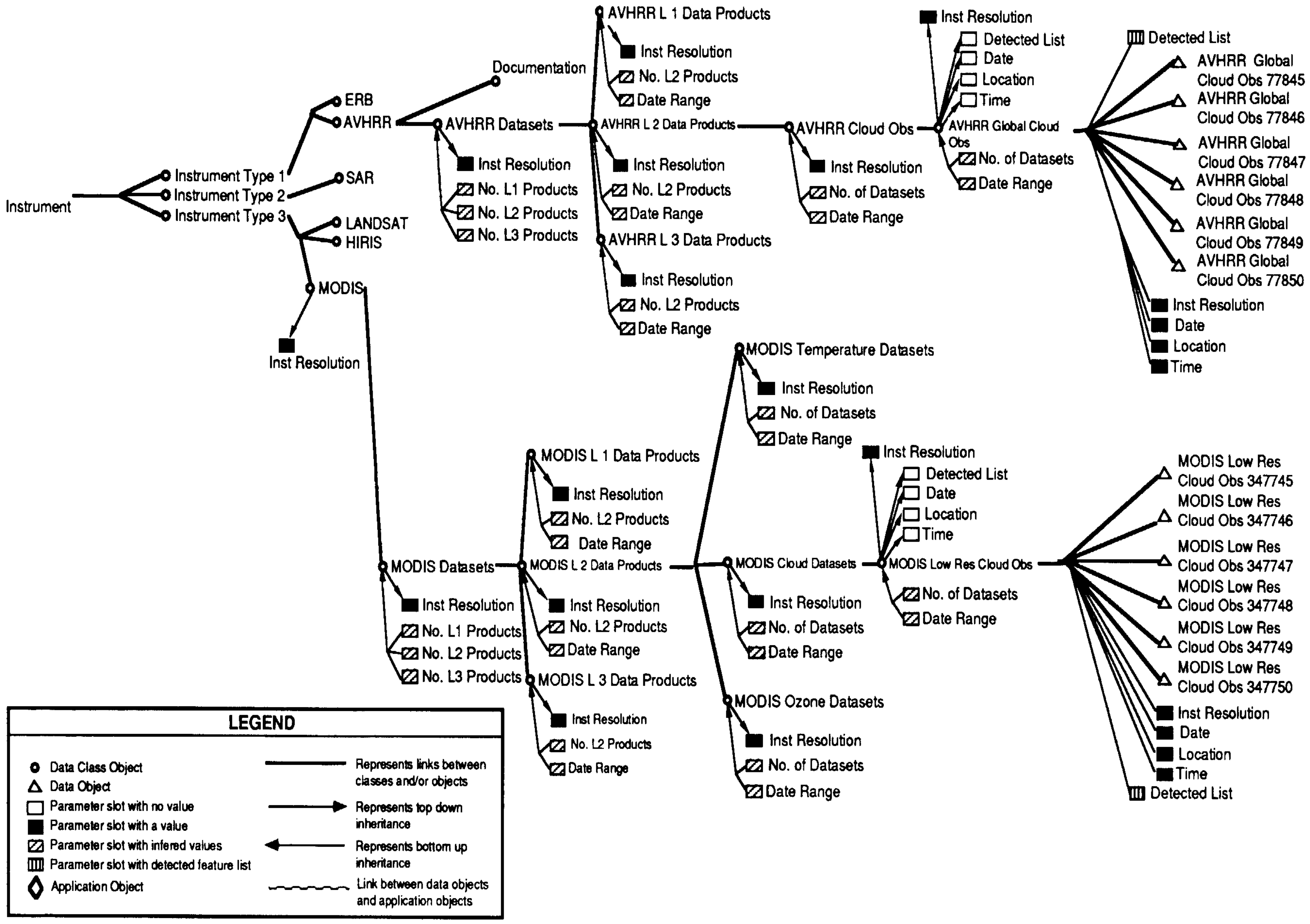


Figure 4
Instrument Super Class Domain

The modeling of the first domain resulted in the representation of the *Area of Interest* super class and the *Parameter* super class discussed in Step 1. The representation of both these super classes is incomplete, with only sufficient detail to demonstrate the semantic modeling concept and how one accesses data by either interest area or parameter.

The actual structure for each of the super classes was based on interviews with a atmospheric scientist supporting the NASA Climate Data System, and was organized to focus on selecting desired sensor observations that were ingested in the instrument super class domain. Figure 5 provide a diagram of this domain.

The representation in atmospheric science focused specifically on clouds and cloud related phenomena. Access to data using the two class objects was supported by links between objects that span between the operational view and the application views. These links are shown in Figure 6, Semantic Model Overall Architecture. The links provide not only the pointer to where a particular data object class can be found in the instrument super class domain, but it also provide a path for meta data to move so that objects that relate to phenomena of interest can be updated with parameter data from the instrument class objects. For example, summary data from the class objects *AVHRR cloud observations* and *MODIS cloud data sets* were linked to the object *Formations* . [1] Within these objects, meta information was further summarized using rules such as would be required to determine the date range for all observations regarding cloud formations along with information as to a list of sensors that observed identified formations. Using the information stored in the summary data property slots, it would be possible to present to a database user information that would assist in selecting desired observations prior to actually browsing through actual data sets. Information stored in a summary property slots might include what instruments are available, the time duration of a sensing activity and the number of observations made.

The modeling of the second domain is based on creating a logical structure of related data classes and objects that can be associated to a specific area of interest, discipline or parameter. The rationale for building this part of the model was motivated by having to support some sort of Automated Data Cataloging and Characterization operation. Basically such an operation would scan a sensor observation for any unique feature and then note this feature in a identified list that is related to the observation itself. Because automated data ingest is still in its formative stages, several assumptions had to be made as to how it will function. The assumptions are:

1. All detected features/objects are placed in a property slot called *detected list* that is associated with the sensor observation object in the instrument super class domain.

2. Detected features are coded so they can be sorted by class/group in some sort of logical structure.

3. All detected features/objects retain parameters that point back to the sensor observation where the feature/object was found.

Given the above conditions, features and objects move from the detected list parameter slot to the science object domain where sets of rules are used to sort the objects into the various grouping where they will be stored and then summarized at higher levels. This part of the semantic data

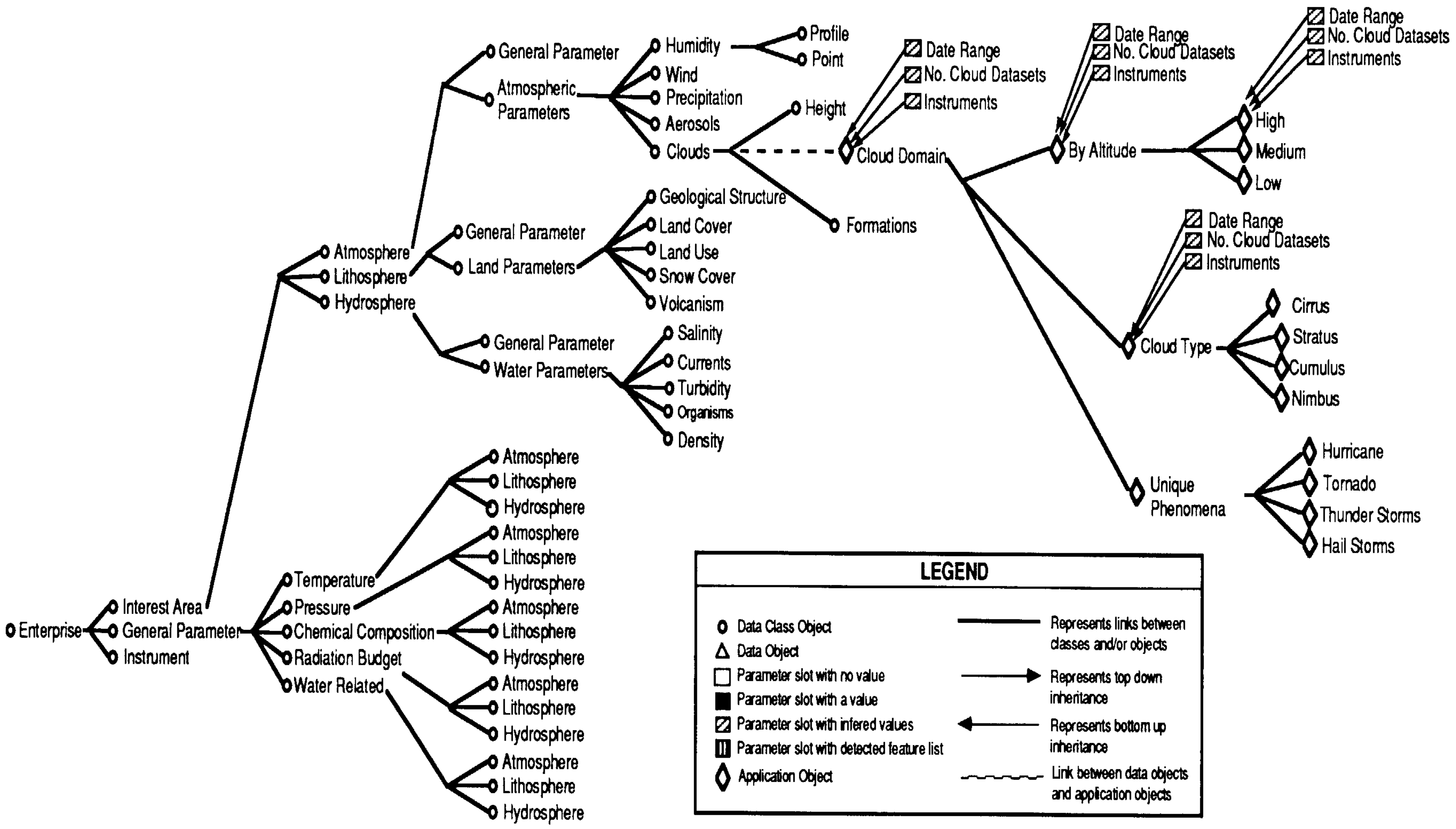


Figure 5
Climate Class Domain

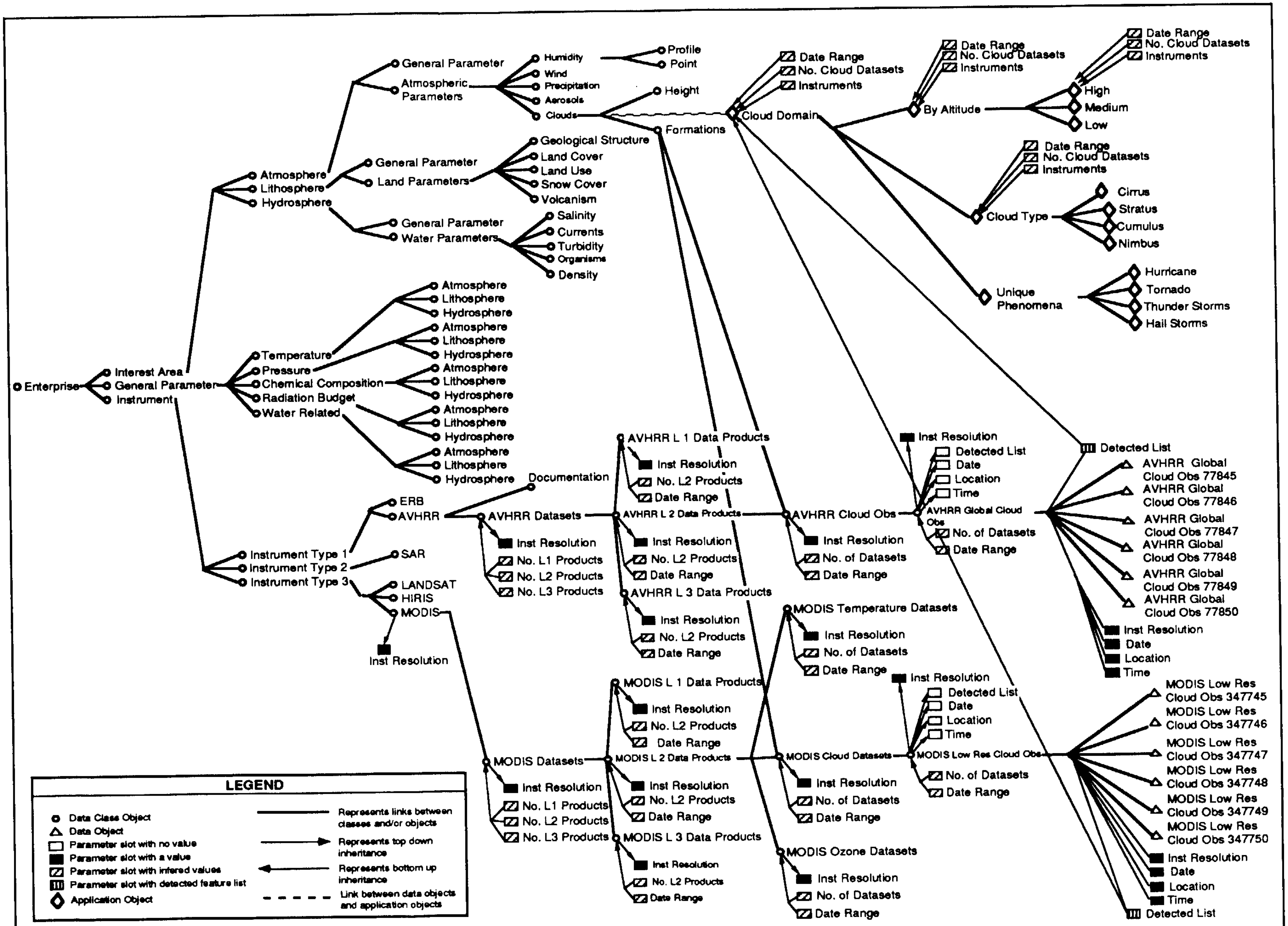


Figure 6
Semantic Model Overall Architecture

model was not prototyped in NEXPERT because of the uncertainties involved in how the object will be ingested and classified. Once these two issues have been resolved then the model can be updated to accommodate the feature/object ingest function.

Conclusions

Once the model prototype was completed, dummy data set objects, and associated detected lists were put into the model and evaluated. The evaluation found that semantic modeling significantly aided in understanding the data domain; something that needs to be done prior to building any database system. Of special note is the fact that semantic data modeling, when coupled with supporting meta knowledge provides a powerful tool for defining the logical architecture, and resultant information system design, for a large mass storage system. In addition, the semantic data modeling design approach provides a method for laying out a information system design and development effort and then a way of measuring the effort's success against some logical standards. Given this, we feel that any future data system design of any size and complexity should consider this approach since it probably is the only way one can be certain that the resultant design is logical sound and supports not only the needs of the database administrator but also the needs of the user.

The design and development of any useful semantic data model will require a great deal of effort that is directly dependent on how complex of the operational and application views of the data domain. It cannot be over emphasized how much effort (much of which probably will be of a high quality like a knowledge engineer) such a modeling activity may require especially for a large mass store system.

Finally, it appears that it is possible to implement a semantic data model with an associated knowledge base using an object oriented expert system development tool like NEXPERT. However, one should be cautioned that familiarity with the tool is quite important as the complexity of the model will dependent on how well the tool is able to represent the domain.

REFERENCES

[1] Earth Observing System, 1989 Reference Handbook, National Aeronautics and Space Administration, Goddard Space Flight Center

[2] W.J. Campbell, L. H. Roelofs, N.M. Short Jr., *The Development of an Intelligent User Interface for NASA's Scientific Databases*, Telematic and Informatics Vol. 3 No. 3 pp 177-190, 1986

[3] E.F. Codd, *A relational model for large shared data banks,* CACM, Vol. 13 No.6 377-387 1970.

[4] D. Tsichritzis and F. Lochovsky, *Hierarchical database management: a survey,* ACM Computer Surveys, Vol. 8 No. 1, 105-124, March 1976.

[5] R.W. Taylor and R.L. Frank, *CODASYL database management systems,* ACM Computing Surveys, Vol. 8 No. 1, 67-104, March 1976.

[6] M. Hammer, and D. Mcleod, *On the Architecture of Database Management Systems*, Infotech State-of-the-Art Report on Data Design, 1979

[7] W. Kent, *Limitations of Record-Based Information Models*, ACM Transactions on Database Systems, Vol. 4, No. 1, March 1979 pp 107-131.

[8] J. Peckham and F. Marynaski, *Semantic Data Models*, ACM Computer Survey, Vol. 20, No. 3, September 1988, pp 153-221.

[9] E.F. Codd, *Further Normalization of the Database Relational Model,* Database Systems, ed. R Rustin. Englewood Cliffs, N.J., Prentice-Hall Inc., 1971.

[10] W. J. Campbell, L. H. Roelofs, M. Goldberg, *Automated Cataloging and Characterization of Space-Derived Data*, Telematics and Informatics, Vol. 5, No. 3, pp 279-288, 1988.

[11] McLeod, D., *A Semantic Database Model and Its Associated Structured User Interface*, Technical Report, MIT Laboratory for Computer Science, Cambridge, Mass., 1978

[12] W.D. Potter, R.P. Trueblood, C.M. Eastman, *Hyper-semantic data modeling*, Data & Knowledge Engineering, North Holland, Volume 4, Number 1, July 1989.

[13] D. Tsichritzis and F. Lochovsky, *Data Models*, Prentice Hall, Englewood Cliffs, NJ, 1982

[14] W. Kent, *Limitations of record based information models,* ACM Trans. on Database Systems, Vol. 4 No. 1, 107-131, March 1979.

[15] J.R. Abrial, *Data Semantics in Data Base Management,* J.W. Klimbie and K.L Koffeman Eds, 1-59, North Holland, Amsterdam, 1974.

[16] R. Hull and R King, *Semantic database modeling: survey, applications and research issues,* ACM Computing Surveys, Vol. 12 No. 1, March 1987.

[17] J.M. Smith and D.C.P. Smith, *Database abstraction: aggregation and generalization,* ACM Trans. on Database Systems, Vol. 2 No. 2, 105-133, June 1977.

[18] N.M. Short Jr, W.J. Campbell, L.H. Roelofs, and Scott L. Wattawa, *The Crustal Dynamics Intelligent User Interface Anthology*," NASA TM 100693, October, 1987.

[19] W.D. Potter, R.P. Trueblood and E.M. Eastman, *Hyper-semantic data modeling*, Data & Knowledge Engineering, Vol. 4, No. 1, July 1989, pp 69-90

[20] W.D. Potter and L. Kershberg, *A unified approach to modeling knowledge and data*, Proc. IFIP TC2, conf. Proceeding on Knowledge and Data, (DS-2), Algarve, Portugal, November 1986.

[21] Neuron Data Inc., NEXPERT *Object Fundamentals*, 444 High Street, Palo Alto, CA 94301

N90-22316

KNOWLEDGE STRUCTURE REPRESENTATION AND AUTOMATED UPDATES IN INTELLIGENT INFORMATION MANAGEMENT SYSTEMS

Stephen M. Corey
Richard S. Carnahan, Jr.

Martin Marietta Information & Communications Systems
Denver, Colorado

1.0 ABSTRACT

Work reported in this paper is part of a continuing effort to apply rapid prototyping and Artificial Intelligence techniques to problems associated with projected Space Station-era information management systems. In particular, timely updating of the various databases and knowledge structures within our proposed intelligent information management system (IIMS) is critical to support decision making processes. Because of the significantly large amounts of data entering the IIMS on a daily basis, information updates will need to be automatically performed with some systems requiring that data be incorporated and made available to users within a few hours. Meeting these demands depends first, on the design and implementation of information structures that are easily modified and expanded, and second, on the incorporation of intelligent automated update techniques that will allow meaningful information relationships to be established. This paper examines potential techniques for developing such an automated update capability and examines IIMS update requirements in light of results obtained from our IIMS prototyping effort.

2.0 INTRODUCTION

The advent of large, information intensive data systems will require sophisticated user access capabilities. In particular, projected large data volumes necessitate implementation of extensive browsing and querying facilities. Indeed, given the prospect of multiterabyte databases (which will be continually updated), it is our contention that without the availability of higher-level information to help focus a user's search, data access would be virtually impossible. To provide rapid access in a data environment whose structure can be transparently and dynamically altered, we employ the conceptual structure of *metadata*, or information about the data, as the main information structure. Additionally, to allow for maximum efficiency in the incorporation of new information, *metadata* structures must be automatically generated and maintained. When an update message is received indicating the arrival of new data, the message is processed inferring new *metadata* that is then appropriately linked to already existing *metadata* in the knowledge structure. As we have mentioned before (Carnahan, Corey, & Snow, 1989), the speed at which this process can be accomplished depends on the type of data received and its potential relationships to other *metadata*. Following sections describe the approaches we have taken to better define requirements for an advanced information management *metadata* knowledge structure and automated update system.

3.0 TYPICAL QUERY FORMULATION

In typical query systems, much is required of the user. Before users can formulate queries to locate data sets related to their areas of interest, a mathematically oriented query language must generally be learned. In addition, users must become familiar with the physical or logical view of the data structure. In a relational database management system (DBMS), this process requires learning attribute and relation names, as well as relationships between them. While such a condition is acceptable for small systems containing few relations and attributes, the situation rapidly becomes unmanageable for very large database systems.

Recently, natural language (NL) query formulation systems have been implemented to aid in the elimination of the need for a mathematical query system. Nevertheless, even though the query language associated with NL systems is not mathematical, the user is still required to learn a query language of sorts. As a result, words and phrases recognized by the NL system grammar and appropriate procedures for combining them must be learned. Unfortunately, users must face other problems when using a NL system. NL systems must be trained for a specific domain under which it will operate. Domain terminology is taught to the grammar so that a specific user may converse in a manner common to that domain. Terminology, however, is often dissimilar among scientific domains; NL systems are generally not well suited for multidisciplinary information. Yet it must be remembered that even if problems associated with different domain terminology are resolved, the user is still required to learn the underlying data structure and its relationships.

With either the NL or the typical query system, the user forms queries without complete knowledge of data contained in the database. In this type of an environment, results of the query may contain data sets of no interest, or the results may contain only some relevant data sets. The former consequence results from either inadequate or inappropriate types of constraints being placed on the query. Hence, the querying process is not completed and the user has to sift through inappropriate data to locate data sets of interest. The latter consequence is more serious; some of the data sets of interest cannot be found because of constraints placed on the query by a user who does not fully comprehend the parameters of the targeted data sets. Furthermore, a null result is also possible and the user is left to wonder at the cause for lack of system response. Database research applied to this issue has resulted in prototype systems in which the user is provided with an understanding of which subclause(s) of the query has caused the inappropriate elimination of data sets, but commercially available DBMSs have not yet implemented this feature. Yet, even when such features do become available, information provided concerning the failed query is *post hoc*; no real aid is provided to the user while the query is being formed. For users who are unfamiliar with the data, the task of forming an appropriate query will be formidable. In light of problems associated with current systems, we have taken a different approach.

4.0 IIMS QUERY FORMULATION

To address the limitations of typical query systems and notably, those encountered when accessing extremely large databases, the query system of the prototype Intelligent Information Management System (IIMS) has been implemented using a *metadata* base (rather than a database containing actual data), and with a querying approach called *assisted query formulation.*

Since databases in which data sets are to be located will likely be extremely large, geographically distributed, and heterogeneous, the possibility of providing a real-time interface with standard techniques is, at best, remote. One way to address this problem is to employ a method that reduces the amount of information to be searched. A typical information reduction technique used in standard database systems is the use of indices. We believe, however, that the exclusive use of this technique will not adequately reduce access time in the type of query environment we envision. In contrast, the IIMS achieves information reduction through the appropriate selection of a small abstraction of all possible information contained in the databases; this abstraction of data we refer to as the *metadata* base. Through the use of a *metadata* base, not only is the amount of information to be accessed reduced, but relationship information among the data can also be included; a capability that would be impossible without abstraction due to overhead associated with this information. Thus, the *metadata* base represents not simply an abstraction of information but a knowledge base of *metadata* and *metadata* relationships, and provides the user with more extensive help to eliminate irrelevant information and increases the probability that all data sets of interest are located.

While the selection of a *metadata* base as the focus of access only serves to bring the problem of very large database access to a manageable level, it does nothing to address query problems encountered in normal systems. It seems apparent that a query system must be created that takes advantage of knowledge contained in the *metadata* base. With *assisted query formulation*, the user is guided through the process of formulating queries on the *metadata* base through menus that present only relevant information. Items displayed to the user are controlled by the underlying knowledge structure and are determined largely by the user's navigation path through the *metadata.* The user is never presented with selections that are not contained in the *metadata* base. Since query formulation is based entirely on the user's navigation path and selections, only valid queries can be formed.

This approach to navigation and query formulation frees the user from having to learn a query language since the knowledge structure provides the user with appropriate next selections that are syntactically and semantically valid. In addition, because the query system presents the user with value information contained within the database, the user never has to guess what an appropriate value is for attributes being presented. For example, when the user chooses the "Programs" *metadata* concept, valid values for this node in the knowledge structure (data collection programs) are presented as selections. The user does not have to be concerned about the form of the query input, whether it is a string, if it contains spaces or underscores, and the like. All valid potential query inputs are presented to the user in a menu that allows the desired item to be selected. Since such information is presented to the user as the query is being formed, the user is actually browsing information contained in the *metadata* base and forming the query based on actual information; as a result, a more accurate query is formed. This process reduces the number of probing queries users have to form before information for which they are searching is located. Furthermore, response time for each step in the query formulation process is rapid due to the fact that only a small amount of *metadata* is being processed at a given time.

Obviously, since projected databases are likely to be very large, not all value information can be contained in the knowledge structure; if so, the size of the *metadata* base would approach or exceed the size of the original databases. However, it is important for the *metadata* base to be able to contain more detailed information in some areas than in others. As a result, it is necessary for the knowledge structure to be able to handle multiple levels of *metadata* abstraction. During navigation, the user will be presented with value information only when available, and in a seamless, transparent manner.

Assisted query formulation also frees the user from having to understand the physical organization of information contained in the *metadata* base. Appropriate information is presented to the user when required and all the user must do is select from presented items. However, the user is not totally released from understanding information categorization. Before the user can select an item, the relative position of the item within the knowledge structure must be understood. Therefore, the user may know the concept (node) in the knowledge structure at which he desires to be positioned, but he may not be able to readily locate or navigate to the concept. To help resolve this problem, the IIMS prototype employs two features: first is the use of domain specific terminology and information organization; that is, information is presented to the user in a familiar manner and is achieved through the use of data views. Second, we have implemented a FIND capability that allows the user to select from a list of appropriate concepts. Only those concepts relevant to the user's interests are presented. Once the selection is made, the IIMS prototype locates the shortest path to the chosen concept from the user's current knowledge structure location and then automatically navigates to that concept. In the future, other navigation assistance tools addressing this issue will also be studied.

5.0 KNOWLEDGE-NET - THE IIMS KNOWLEDGE MODEL

To provide the types of capabilities described above and still satisfy the critical requirements of dynamic modification and update (which in most instances, we believe, will have to be performed automatically) we have implemented *metadata* knowledge and relationship information using a data representation rather than procedural approach. The data representation we are using can be categorized as a semantic network consisting of typed nodes and typed, unidirectional links. One unique feature of our implementation is that relationships have conditional relevance, that is, not all links emanating from a particular node are relevant under all conditions. Link relevance is computed dynamically and is based primarily on the navigation path traversed by the user through the knowledge structure. Determination of link relevance at any given time and its use in the query formulation process is discussed in section 5.5.

In the knowledge-net (k-net), *metadata* concepts or facts are represented as typed nodes while relationships between concepts are represented as typed links. Node or link characteristics and their effects on knowledge structure composition and use are governed by their types. In the current system, eight types of nodes (*Fact, Information, ISA-S, ISA-V, Paren, Operator, Structure, Value*) and ten types of links (*Abstraction, Acronym, Alias, Fact, Information, ISA, Reverse, Structure, Value, Value Selection*) are used. Of course, the number of types of both nodes and links is not fixed and will likely change as the k-net increases its expressive capabilities. Rather than discussing separately each of the typed nodes and links listed above, each will be discussed, whenever possible, in the context of its combined use with other nodes and links to represent knowledge. Primary knowledge representation capabilities of the k-net are implemented using *Fact, ISA-S, ISA-V, Structure* and *Value* nodes and *Abstraction, Fact, ISA, Structure, Value,* and *Value Selection* links. Other node and link types are generally used to define associated information, alternative names for domain concepts, and syntax representation (see section 4.2). Figure 1 illustrates a sample *value tree* from the current IIMS knowledge structure.

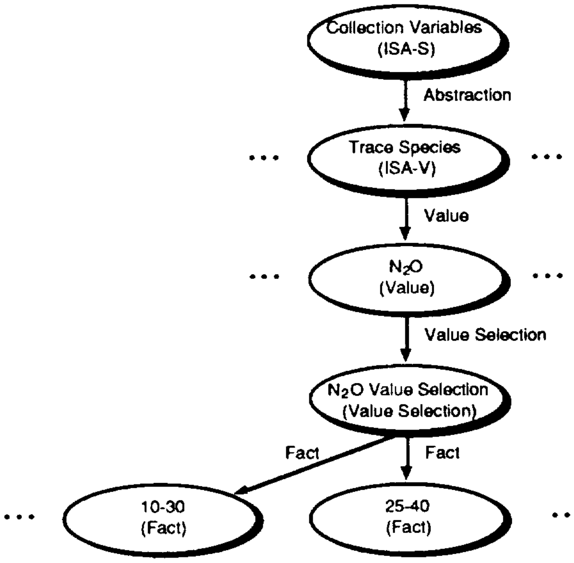


Figure 1 Example Knowledge Structure Value Tree

The current implementation of the k-net may be thought of as a series of *value trees* grouped together through the use of *Structure* nodes (while this is generally true, it should be noted that the knowledge structure is highly interconnected). Similar to any typical hierarchical representation of knowledge, *Structure* nodes are generally used as category headings. Similar to a directory structure, they are also used to represent a concept which does not directly have an associated value. Selection of structure nodes does not cause a change in the status of the query currently being formed and hence are not involved in any value processing performed.

5.1 VALUE TREES - ATTRIBUTE/VALUE BINDINGS

Within the k-net, *value trees* control the setting of attribute values (attributes are parameters describing appropriate data sets). Query formation is fundamentally the process of setting values or ranges of values for known data set attributes and relationships among attributes. This process is achieved by navigating through relevant nodes of the k-net. However, no constraints are added to the query until entering a value tree and indicating the value for a specified attribute. Of course, this process is transparent to the user and navigation procedures are not perceptively different when a value tree is entered. *ISA-S* nodes are the attribute nodes of the k-net, and are the only nodes that can be assigned a value. Values of *ISA-S* nodes are designated by nodes below them in the tree and hence, they represent root nodes of small *value trees* embedded within the k-net. *Value trees* are generally never more than a few levels deep, but node values may be set at different levels within the *value tree*. This is caused by the iterative refinement of the original domain concept into more specific detail represented by the *value tree*. For example, in the current prototype k-net, the node "Collection Variables" is an *ISA-S* node which has "Trace Species" as one of the concept nodes below it (see Figure 1). The "Trace Species" node represents a valid value for the "Collection Variables" node whose value specification, at this point, can be terminated. However, if "Trace Species" is still too general, the user can select the "Trace Species" node and more specific concept nodes or values of "Trace Species" will be offered to the user. Upon selection of one of these more specific concept nodes, the value for the node "Collection Variables" is altered to reflect the new selection. A *Value* link is used to model a value relationship between two nodes, in which the destination node represents a legitimate value of the originator node. This relationship implies that the destination node represents an *ISA* type of the originator node. Because of this, whenever a *Value* link is defined, an *ISA* link is automatically defined in the opposite direction of the *Value* link. If the destination node of the *ISA* link (i.e., the originator node of the *Value* link) is a *Structure* or a *Value* node, the node type is altered to indicate that this node now represents an *ISA* concept. When a node type change is required, a *Structure* node is changed to an *ISA-S* node and a *Value* node is changed to an *ISA-V* node.

When the query formulation process arrives at the "Collection Variables" node, the root node of the example *value tree* illustrated in Figure 1, determining what concepts to present to the user is governed by the *ISA* concept structure mentioned above. Determining whether the concept "Trace Species" should be presented does <u>not</u> depend on the relevance of the link between "Collection Variables" and "Trace Species" but rather on the relevance of the links between "Trace Species" and "N_2O" and other related concepts. To represent this correctly, all RELEVANCE conditions (see section 5.5 for a discussion of RELEVANCE conditions) on links emanating from "Trace Species" would have to be associated with the link between "Collection Variables" and "Trace Species". Such a procedure is inefficient since multiple copies of the same relationship representation must be maintained. Instead of this approach, we have chosen to implement an *Abstraction* link, representing the notion that the relationship between nodes is an abstraction of the value of the node attribute being set.

Upon encountering an *Abstraction* link, the system progresses directly to the appropriate destination node and begins relevance processing on links emanating from that node. When one of these *Value* links is found to be relevant, given the current state of the k-net, the *Abstraction* link is then considered to be relevant. Abstraction relevance processing is an iterative process. If links from the destination node of the *Abstraction* link also contain *Abstraction* links, relevance processing then shifts to the destination node of the new *Abstraction* link. The process continues until no *Abstraction* links are encountered, at which point relevance processing then begins. Under appropriate conditions, *Abstraction* links, like the *ISA* link, are generated automatically whenever a *Value* link is defined. Figure 2 illustrates the state of the *value tree* at the time a request is received to define the *Value* link between "Trace Species" and "N_2O". Since the *Value* link between "Collection Variables" and "Trace Species" represents the same relationship as the *Abstraction* link is intended to represent (see Figure 3), the *Value* link is deleted from the k-net and replaced with the *Abstraction* link. Thus, defining a *Value* link can generate three links, delete one link, and change a node type. Since relevance of the *Abstraction* link is determined by other links, *Abstraction* links are allowed only to contain view relevance and are processed using *view filtering* (see section 5.5).

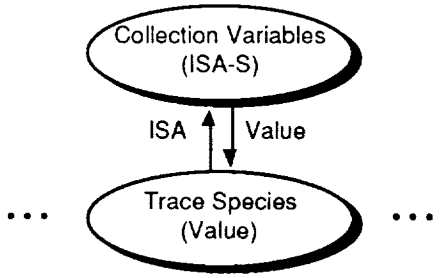


Figure 2 Value Tree State Prior to "Trace Species" Value Link Definition

Originally, *ISA-S* nodes are specified as *Structure* nodes but their type is changed to *ISA-S* when an *ISA* link creation request is received specifying the *Structure* node as the destination node of the link. Specification of the *ISA* link indicates that the *ISA-S* node represents a node value abstraction and, thus, has different characteristics than a typical *Structure* node.

When an *ISA* link is defined, an *Abstraction* link is automatically defined in the opposite direction. Determining the relevance (see section 5.5 for a discussion of relevance processing) of the *Abstraction* link should be based on the relevance of the links to the concept which is the destination of the *Abstraction* link. The *Abstraction* link is considered relevant if at least one of the value links of its destination node is relevant. *Abstraction* links are used to prevent maintaining duplicate copies of relevance instances and the situation where an *Abstraction* concept node is offered to the user, but upon its selection all further selections have been eliminated because of relevance processing. For example, the concept "Collection Variables" has an *Abstraction* link to the node "Trace Species". When the k-net determines if the node "Trace Species" should be offered, value links emanating from "Trace Species" are relevance processed. Upon locating the first value link from "Trace Species" considered relevant, the link from "Collection Variables" to "Trace Species" is then declared relevant. Relevance processing is recursive and if any links emanating from "Trace Species" are also *Abstraction* links, each destination node of each Abstraction link is processed in turn until a relevant link is located. As stated above, the node "Trace Species" is a valid value selection for the *ISA-S* node "Collection Variables". However, it is also an abstraction of more specific value concepts. This type of node, one which represents a value but is not a leaf value node, is referred to as an *ISA-V* node. As with *ISA-S* nodes, the *ISA-V* node began as a *Value* node but was modified when a request was received to define an *ISA* link with the *Value* node as the destination.

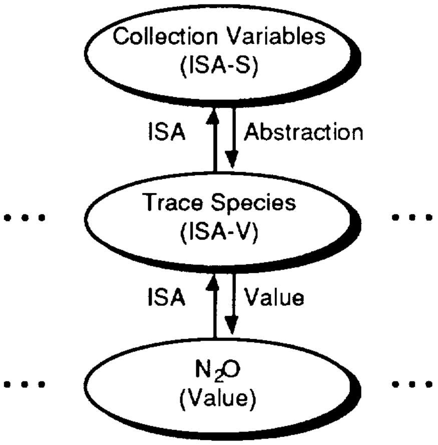


Figure 3 Value Tree State After "Trace Species" Value Link Definition

Leaf values of *value trees* are generally *Value* nodes, although there are exceptions. *Value* nodes represent a valid value for the attribute node that is the root of the tree. They represent the most specific domain concept known for any attribute and, hence, do not have links to more specific concepts. The only exception to this rule is when value range type information is known. In such a case a *Value-Selection* link is defined for the node pointing to an appropriate *Structure* node. The *Structure* node then has *Fact* links defined that point to *Fact* nodes representing specific ranges of values for the associated *Structure* node. Upon selection of the *Structure* node, a routine is invoked that displays relevant value range information, allowing the user to indicate value ranges of interest. In this way, the *value tree* structure allows multiple levels of data abstraction with different *metadata* items having varying levels of attribute value abstraction.

Although the node value structure has been described in terms of a tree (which it appears to be upon first glance), there is nothing to prevent links from entering the structure at any location from outside, or to prevent links within the value tree from pointing outside the structure. The latter condition is almost always the case although these types of links are generally not involved with the selection of a value for an attribute, but are links to other related concepts. It is important to realize that the *value tree* structure allows setting attributes to desired values, with only relevant and valid values being offered to the user. Thus, the user browses relevant data while forming the query.

As stated above, the k-net may be thought of as a series of *value trees* grouped together through the use of *Structure* nodes and *Structure* links and representing category headings or other structural components. While defining a *Value* link causes a companion *ISA* link to be defined in the reverse direction of the *Value* link, not all links cause the creation of companion links. However, certain analysis procedures require traversing structural and value relationships within the k-net. Therefore, to provide this capability, a *Reverse* link must be defined in the opposite direction of the *Structure* link. It is important to understand that *Reverse* links are not used to model relationships in the k-net and are not used in any way to control the navigation process; they are only used to provide for required analysis capabilities.

5.2 SYNTACTIC REPRESENTATION

While assigning values to attributes is a large part of the query formulation process, it is also necessary for these bindings to be grouped and joined in a logical manner. *Paren* and *Operator* nodes are used to implement this capability for English-like syntax representation in the k-net, and these nodes have inherent capabilities that impact query syntax. For example, a right paren cannot be offered unless there is a corresponding open left paren, or a conjunction cannot be offered until an attribute/value binding is completed.

5.3 NON-QUERY INFORMATION REPRESENTATION

Information nodes and links provide the capability to provide non-query related information about nodes to which the user has navigated. Information nodes are usually entry points into the tutorial system and provide such information as drawings, explanations, and technical specifications. In the IIMS prototype, this type of information is provided by hypermedia presentation so the user can use the entry point as the beginning location for an in-depth exploration of the given subject, or related subjects. Selection of these nodes does not directly add anything to the query.

5.4 ALTERNATIVE NAME REPRESENTATION

Two links are used within the k-net to represent alternative names for nodes: *Acronym* and *Alias*. The presence of an *Acronym* link indicates that a certain node has an associated acronym, and, since these links can have relevances associated with them, a single node may have more than one acronym. In the IIMS prototype, a user can request that a node be displayed using either its full name or its acronym. If acronym is selected, the IIMS prototype processes relevance instances associated with each *Acronym* link until one is found to be relevant. The acronym name given to the destination node will then be used as the display name of the node.

The *Alias* link is used to allow alternative names for nodes. Such a situation is made necessary given the condition where one node represents a single concept but different domains may have different names for that concept. Obviously, this link is used only when the concepts are exactly the same. If concepts are slightly different, a different node will have to be defined.

5.5 LINK RELEVANCE PROCESSING

Within the k-net, a single concept (i.e., node) appears only once within the semantic network. However, during the navigation process, it may seem to the user that many nodes exist expressing the same concept. Such a perception is caused by choices the user is offered upon selection of a concept. Remember that choices offered are dependant upon the path the user has taken to arrive at the concept. As briefly discussed above, this is achieved through the use of relevance descriptions associated with each link. Therefore, when a user arrives at a node, the k-net processes each link emanating from the node. Only those links determined to be relevant are used to define the next choices offered to the user. For display purposes, this conditional relevance technique is used to reduce the number of nodes and links typically associated in a highly interconnected network.

Associated with each link is a list of RELEVANCE conditions. When a link creation request is instantiated, the system determines if a similar link-type already exists (i.e., the same as that requested other than the associated RELEVANCE condition). If such a link is found, a new link is not created. Instead, the RELEVANCE condition of the new link request is added to the existing link. As a result, unbridled proliferation of links is controlled. Currently, two types of RELEVANCE conditions are used: COMPLIANCE and NON-DISPUTING.

If a RELEVANCE condition is specified as COMPLIANCE and the condition fails to be satisfied, the link is rejected without further processing. However, if a RELEVANCE condition is NON-DISPUTING, then as long as the link's relevance is not disputed by the current state of the query (i.e., a variable set by the query process does not have a value conflicting with that specified in the RELEVANCE condition), then failed relevance does not cause the processing of the link to be terminated. Such a process is made necessary to allow specification of RELEVANCE conditions that will not cause rejection of the link if variables used in the specification of the RELEVANCE condition have not been set by the query process. For example, if a RELEVANCE condition is specified by the fact that the program must be NIMBUS and a NON-DISPUTING condition is present, then the link's relevance is rejected only if the program is specified to be something other than NIMBUS. If the program variable has not already been set, link relevance is not rejected. However, if the RELEVANCE condition is COMPLIANCE, then the link's relevance is accepted only if the "Programs" variable is set to NIMBUS.

Determination of link relevance also involves processing the non-reentrant state (see section 5.5.1) of the link's destination node and then processing RELEVANCE conditions associated with it. If the link passes the non-reentrant filter test, then RELEVANCE conditions associated with the link are processed in two steps: *view filtering* and *relevance predicate processing*. Results of this two-step process can be used to immediately accept or reject the link or indicate that no information concerning link relevancy can be provided. In the second case, the next RELEVANCE condition is processed. This process continues until the link has either been accepted or rejected, or there are no more RELEVANCE conditions to process; in such a case the link is accepted as relevant. The second process step is more involved and takes more time than the first. In this way, *view filtering* and *non-reentrant processing* act as rapid filters that eliminate many links from the more intensive and time consuming *relevance predicate processing*. Each of these processes are discussed more fully in the following sections.

5.5.1 Non-Reentrant Nodes

It is necessary that the k-net be able to represent the fact that under certain conditions some nodes cannot be reentered during the navigation process. In the current k-net, most nodes representing variable binding may not be reentered under an AND condition. This requirement can be illustrated when one considers the possibility of a single data set having two Analysis Product Qualities: A AND B; obviously, such a state is untenable. However, it is possible for a data set to have an Analysis Product Quality of A OR B. Therefore, the node "Analysis Product Quality" must have non-reentrant capability so that it will not again be offered to the user if it has already been offered within the scope of the current AND condition, but may again be offered under other conditions. Non-reentrant capabilities have been implemented in the current k-net, and checking this condition is the first step in link relevance processing, needing to be performed only once per link. If the destination node of the processed link indicates that the non-reentrant attribute is activated, syntax of the current state of the query formed through navigation is analyzed to determine if a restrictive condition exists. If so, the link is rejected and processing on this link ceases. If the link passes the *non-reentrant processing* test, individual RELEVANCE conditions are processed until determination can be made about the relevance of the link. The first step in this process is passing the RELEVANCE condition through a *view filter* and then, if successful, relevance processing concludes with *relevance predicate processing.*

5.5.2 View Filtering

A view in the IIMS prototype is a set of nodes and links which may be composed of, but are not limited to, nodes and links of the system-wide knowledge structure. Each view is the encapsulation of knowledge relevant to a particular interest or domain and contains domain relevant concepts, relationships, and terminology not included in the system-wide knowledge structure. User views, then, represent sets of base knowledge structure changes required to reflect user-specific domain knowledge. Views provide the user with the capability to form queries in a familiar environment and exclude information irrelevant to interests defined by the view. In the IIMS prototype, views are used to aid in the efficient presentation of relevant information to the user, and to present the information in an appropriate form for the specified user's interests. Separate views defined in the IIMS prototype form a hierarchical inheritance structure in which a view inherits all associated subview modifications. For example, a Physics view might be composed of all concept and relationships defined in the Astrophysics, Geophysics, and Atmospheric Physics views since the generalized Physics concept consists of all these specific domains.

Each RELEVANCE condition contains a list of views for which the associated link is considered relevant. To determine the relevance of the current RELEVANCE condition, this view list must be compared against the view in which the k-net is currently operating. A link passes the *view relevance filter* if one of the views or subviews for which the link is relevant is currently active. To use the example above, if the current view of the k-net was Physics and one of the views of the current RELEVANCE condition is Astrophysics, the RELEVANCE condition would pass the *view relevance filter*. Of course, the converse would not be true. Once a view passes the *view relevance filter*, view filtering of the link can be terminated and relevance processing can continue into the next phase. If no views contained in the RELEVANCE condition are found to be relevant, then processing of the RELEVANCE condition is terminated.

5.5.3 Relevance Processing

Once the link has passed both *view* and *non-reentrant filtering*, it enters the next phase of relevance processing. As discussed above, there are two types of relevance instances: COMPLIANCE and NON-DISPUTING. The RELEVANCE conditions associated with the link are grouped according to these types. All COMPLIANCE relevance conditions must be processed and all must be satisfied. If one of these relevance instances fails, relevance processing of the link ceases and the link is rejected. If all COMPLIANCE tests are completed successfully, NON-DISPUTING relevance conditions are then processed. In this instance, a negative result does not cause the rejection of the link while a positive response results in the link being immediately accepted.

Associated with every RELEVANCE condition is an EVALuable predicate that determines conditions under which the link is relevant and a set of variables that must be set before the predicate is EVALed. Values to which variables will be set are determined from the current state of the k-net. While navigating the k-net, certain attribute/value bindings and relationships between them have been formed. For example, when a user navigates to the "Programs" node, all relevant programs are offered for selection. Once a specific program has been selected, a binding for the node "Programs" has been made. Once value bindings of specified variables have been set based on the current state of the k-net, the relevance predicate is EVALed. The resulting action depends on the type of RELEVANCE condition being processed. It is possible for an attribute to have more than a single value set during the navigation process (through the use of an OR operator) and thus, if the result of the EVALuation of the relevance predicate is NIL, then, maintaining the original semantics of the query, alternative values for the variables are tried and the predicate is reEVALed. This process continues until the predicate returns T or there are no more alternative values for the variables.

Use of the EVALuable predicate introduces an element that is less flexible than desired. Originally, relevance was determined based strictly on the attribute/value bindings established during the query process. An elaborate process that involved "covering" all facts associated with link relevance was used to determine if the current navigation path supported acceptance of the link as relevant to the current state. This technique worked well in most cases since query formation is, basically, the process of indicating desired values of certain attributes describing information to be retrieved. However, this method proved less robust than required for more involved selection criteria such as conditions under which links to the *Paren* nodes are valid. This case involves more than simple attribute/value bindings but rather, involves qualifications of a semantic nature. Therefore, to provide the capability to handle these more complex relationships, the EVALuable predicate expression was employed, making link relevance more difficult to set or change. Yet, with certain tools that we will provide, the problem should be manageable.

This limitation most frequently arises when the user attempts to define his own link. To do this users must specify the conditions under which the link is relevant. By providing the capability to allow the user to input attribute/value bindings and associated conjunctions through the use of a form system, most of the difficulty can be eliminated. This utility will then convert user input into an appropriate predicate to be included with the link. However, for those instances when this procedure will not suffice, the user will be able to directly input the code satisfying the link's relevance requirements. This issue will be studied further, but in any case, original source code will not have to be changed nor will the system have to be rebuilt if RELEVANCE conditions of a node or group of nodes need to be altered. Hence, original requirements placed on the knowledge structure are satisfied.

6.0 AUTOMATED KNOWLEDGE STRUCTURE UPDATES

This section focuses on the primary requirement that future information management systems be able to incorporate new data and information in an automated fashion. This requirement is particularly important when one considers the projected rates at which data will be captured, the projected volumes of data that will be stored, and the secondary requirement[1] that all updates be managed in a completely dynamic environment; that is, the system must be able to accept continuous updates at the same time multiple users are accessing the system. As we have suggested earlier (Carnahan, Corey, & Snow, 1989), several alternatives for implementing automated updating are possible.[2] Following sections describe in detail the initial approach we have taken.

6.1 AUTOMATED UPDATE REQUIREMENTS

As we have already suggested, the requirement to automatically update data, as well as information about the data, will drive the design and implementation of advanced information management systems. The approach we have taken to the issue of automated data/information updates is based on the premise that any future system will have to consider not only which, if any, existing data parameters to classify as *metadata*, but also how those parameters may be translated into more meaningful information related to other, existing information.[3] Obviously, the second objective, to relate information, is of paramount importance if the user is to have any success at all in accessing the large volumes of available data. Figure 4 shows our functional approach to IIMS implementation. Those components we believe are best suited to the application of intelligent systems are indicated. For purposes of the discussion at hand, we will focus on the three components comprising the *Information Update System.*

In general, we view the process of updating information about data to be two-phased. The first phase includes processing of the update message by the *Knowledge Encapsulation System (KES).* Described in more detail in section 6.2, KES functionality includes the capability to infer *metadata* information from data parameters provided by the update message, and then relate that information to already existing information resident in the *Knowledge Net.* The *Update Processor* is not only responsible for providing the update message to the KES, but, just as important, it is responsible for translating the output of the KES into something the *Knowledge Net* can understand Neither process, information inference nor translation, is trivial.[4] As discussed

[1] Some may argue its primacy.

[2] The key criterion here is that the user's search efficiency and effectiveness are minimally impacted by the the ability of the update system to provide adequate *metadata* and to establish appropriate links to already existing information.

[3] While the issue of updating the database where the actual data are stored must also be addressed, it is the not the focus of our current work.

[4] It is important to realize, however, that we view the two

earlier, the *Knowledge Net*, or *metadata* base, is that portion of the IIMS where *metadata* information is stored. When users access the IIMS and browse or query the system, they deal almost exclusively with the *Knowledge Net*. The *Knowledge Net* includes procedural information concerning navigation paths and associated dynamic relevance weights used to present users with the most appropriate information given the user's interests and present location within the *Knowledge Net*. The *Knowledge Net* also contains control knowledge necessary to form the appropriate node and link updates once new information has been inferred and then transferred by the *Update Processor*.

6.2 KNOWLEDGE INFERENCE FROM METADATA

Update messages are generated whenever new data sets are input and they serve to inform the system of the various known data set parameters.[5] The vast majority of these new data sets will not add any new knowledge concepts or relationships to the knowledge structure since information known about the data set is an abstraction of the real data. In most cases, this abstract information will not actually include information about data set range values and hence, each data set collected by an instrument during a certain mission phase will only differ from a previous data set by temporal and spatial values.

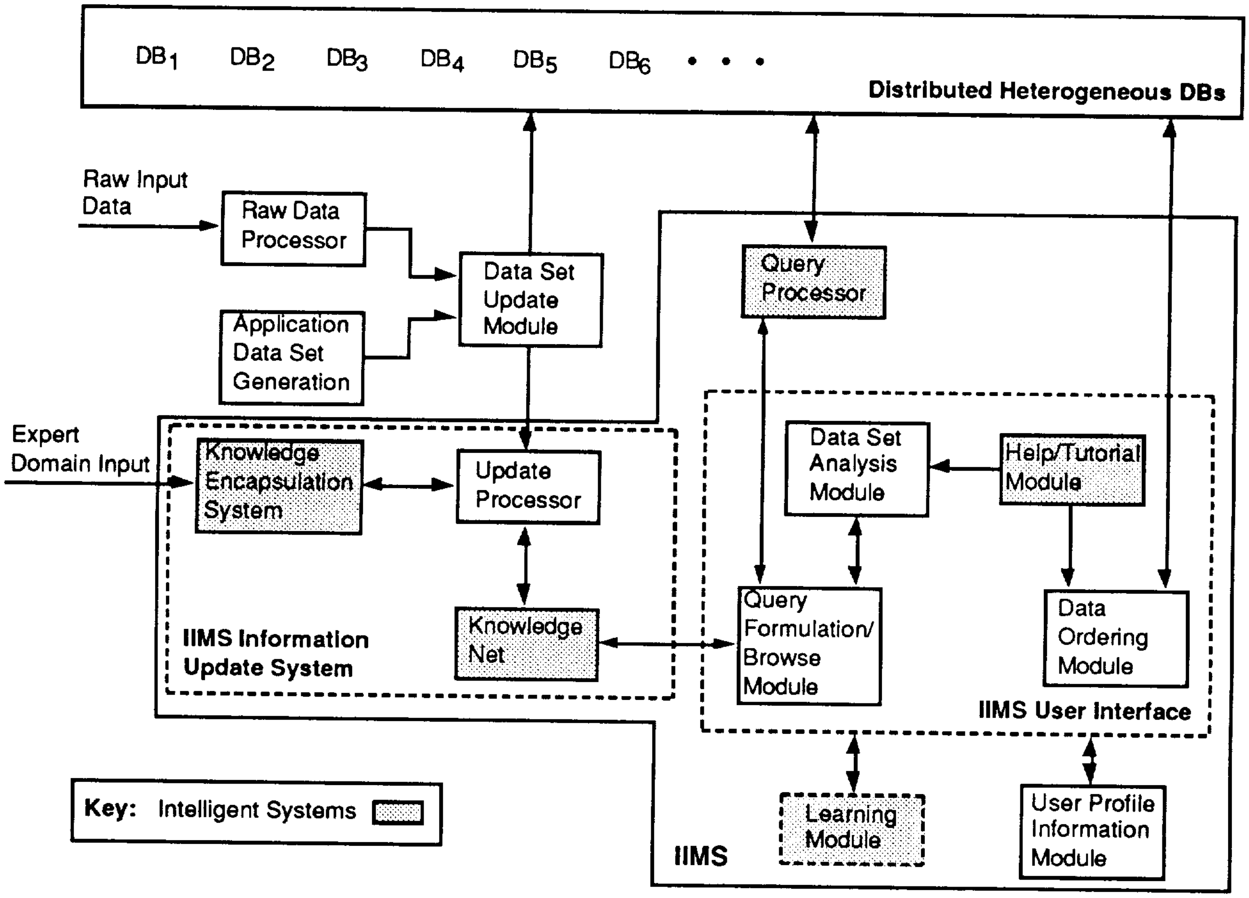


Figure 4 IIMS Implementation Approach

Automated processing of update messages proceeds in two phases. The first phase (*analysis phase*), involves extracting a set of knowledge concepts and relationships represented by the data set parameters existing in the update message. It should be noted that this process involves more than a simple, direct extraction of attribute/value pairs; it also involves reasoning about the data parameters specified in the update message. The second phase (*distribution phase*), takes this reasoned set of knowledge concepts and relationships and integrates it into the basic knowledge structure, as

processes as independent.

[5] As Figure 4 indicates, such data sets could be raw data that has not been processed to any higher level than level 0 or higher level processed data input from data users. In the latter case, we would expect that some level of *metadata* other than basic data set parameters would already exist.

well as into other knowledge used by the system. The *distribution phase* is also responsible for storing data reflecting knowledge structure changes so that user views will be updated appropriately. These two phases are examined in more detail in the following sections.

6.2.1 Inferring Metadata - The Analysis Phase

Analysis of the update message, the first phase of automated update processing, must extract all relevant concept and relationship information contained, either explicitly or implicitly, in the update message. To do this requires the ability to incorporate expert information about relevant scientific domains into the system for use in reasoning about appropriate *metadata* and *metadata* relationships.[6] Information such as data set interrelationships, the relationship of terminology used by different domains, and information concerning the potential relevancy of new concepts and relationships (including the view(s) under which the relationship is valid) must be encapsulated in a form that can be applied to the update message and that can be altered when necessary. An expert system[7] is the most likely choice for this type of knowledge representation as long as the expert system development environment allows for easy modification of the knowledge it contains and can be invoked from a program.

The process of updating the expert system does not have the same stringent restrictions as those placed on updating the knowledge structure since the expert system will only be used to process update messages. Accessing of the knowledge structure will not be impacted if the knowledge encapsulation process needs to be shut down for a period of time to effect modifications to the expert system. The real problem then becomes the ease with which the expert system can be updated and how such updates are distributed.[8]

It is important to note that the set of knowledge concepts and relationships generated by this phase is independent of the knowledge representation scheme used in the actual knowledge structure; if the knowledge structure representation system changes, the module performing the *analysis phase* will not have to be altered. Additionally, the translation of information generated during the *analysis phase* to a form that can be input to the knowledge structure is totally dependent on the knowledge structure implementation. An example of the types of information we expect to be provided from the KES is given in section 6.2.3.

6.2.2 Reasoning and Knowledge Structure Modification - The Distribution Phase

The *distribution phase*, the second phase of automated update processing, uses the set of knowledge concepts and relationships generated by the *analysis phase* to make appropriate modifications in overall system knowledge. This process involves updating (1) the knowledge structure accessed by users, (2) system and user views used in navigation, and (3) other knowledge used by the system.[9]

Five types of knowledge structure updates can be generated from modifications to the expert system or from the *analysis phase*. They include:

1. Adding new relevance conditions to already existing relationships;
2. Adding new relationships;
3. Adding new knowledge concepts;
4. Deleting existing relationships;
5. Deleting existing knowledge concepts.

The first three update types can be generated from either the *analysis phase* or by a modification of the expert system; the last two can only be generated from a modification to the expert system. As stated above, these updates can affect three knowledge areas in the system, the base navigation knowledge structure, the system and user views, and other knowledge structures. The first two will be more fully developed below while the third area has yet to be defined in the current system.

Modification of the base knowledge structure is straightforward and is handled no differently than the normal process of creating knowledge concepts and relationships handled by the knowledge representation scheme. The set of knowledge concepts, appropriate relationship information, and the conditions under which both concepts and relationships are relevant are generated during the *analysis phase*. All that must be done is translate

[6] Obviously, selection of what scientific domains are relevant is somewhat subjective and influenced largely by user interest.

[7] Or more globally, a reasoning system.

[8] One could speculate wildly here concerning the potential applicability of automated knowledge system updating or learning systems. At this time, we believe such updates are likely to be generated manually, at least initially.

[9] Other knowledge might include that used by a learning system to interpret user actions in terms of goals. We can envision, for example, a parallel system that monitors user activity to determine abstract types of information the user may be interested in. Such information could then be incorporated into the user's profile and views to make future sessions more efficient.

this information into the appropriate requests to the knowledge system. Knowledge structure modification requests take the same form as those initially used to create the original knowledge structure; indeed, the same methodology to create new nodes and links is used. Since an initial requirement of the knowledge system was that it must have the ability to be dynamically updated, execution of knowledge structure modification requests can be accomplished with no operations interruption to current users of the system.

The more complex task resulting from updates is the modification of user views. User views are the overlying representation of a set of additional or different knowledge concepts and relationships than those defined by the base knowledge structure. Therefore, user views reflect knowledge concept and relationship change sets applied to the base knowledge structure, creating access environments desired by individual users. When a modification occurs to the base knowledge structure, the change may be outside the domain or set of domains represented by any user view.[10] In such a case, no changes in user views need be reflected. However, it is highly possible that a base knowledge structure modification would affect the knowledge concepts and relationships within any particular user view. In this case, several situations may be encountered. For example, users may not agree with the change or the change request represents an inadequate or inaccurate relationship within a user's domain of interest and hence, it would not be appropriate to include the change in that user's view. However, in most cases, the user will likely want changes to the base knowledge structure reflected in his or her view so that the most recent information is available. If such is the case, nothing special needs to occur. In the first situation, several modifications would be required.

In addition to user view updates, it must be remembered that existence of a significant potential for a large number of user defined views could result in an inordinate amount of time being required to update all user views concurrently. Obviously, such a condition is undesirable. The following methodology is used to address this problem. When an update request is received by the base knowledge structure, the update is logged in a database keyed, for example, on time of request initiation and possibly the potential user view(s) involved. As a result, when a user selects a view, the update database is checked to determine which updates have been executed since the last time the view was instantiated and which of those are applicable within the scope of the view. (Remember, as suggested earlier, most update messages will not involve actual changes to knowledge concepts or relationships defined in the base knowledge structure and hence, the number of updates potentially relevant to the user should be small. Changes reflecting temporal and spatial differences in data sets will not invoke an update resulting in a potential user view modification.)

If modifications affecting the user's view have been logged, the user is notified that potentially relevant changes have occurred. The user then has the option of addressing the changes at the current time or processing them at a later time. When the user decides to address the changes, a list of request changes are presented and the user can select those considered relevant within the current view. When the selection has been made, appropriate changes will be made to the view. If the change is accepted, nothing will have to be done to the user's view since the change has already occurred in the base knowledge structure. However, for those changes that are rejected, appropriate modifications need to be made in the user's view to undo modifications already made to the base knowledge structure. If the user chooses to postpone addressing the changes, the user's view will have to be modified to undo the changes which were made to the base knowledge structure.

Since most users will likely want to accept changes automatically, a facility is provided to allow the user to select a mode of system operation in which changes are immediately instantiated within the appropriate user views. Or, if desired, the user can be prompted for each change before incorporation.

6.2.3 Automated Update Message Processing - An Example

To help clarify the process of automated update message processing and potential difficulties that may be encountered when implementing such a capability, several example update messages are provided.[11] Additionally, further clarification of reasoning required for automated knowledge structure updating is given in terms of potential *analysis phase* output.

The first example update message is taken from an amateur observation of a comet under the jurisdiction of the International Halley Watch organization. Most of the

[10] Such modifications might occur, for example, when new missions are launched and the base knowledge structure is modified to reflect the new information.

[11] Example update messages represent information extracted from the headers of various data files included in the *NASA, Space Science Sampler, Volume 2: PDS Interactive Data Interchange CD-ROM*. These data files represent various data and header formats in common usage in scientific domains. We are assuming that types of information contained in the headers represent the types of information to be included in update messages for processed data sets.

attribute names are extracted directly from the file.

Update Message 0001

```
Object=P/CROMMELIN
File-Num=800200
Date-Obs=29/12/83
Time-Obs=.7600
Date-Rel=13/12/85
Discipln=Amateur
Long-Obs=999/99/99
Lat-Obs=+99/99/99
System=85000000
Spec-Evt=F
Dat-Type=Visual Mag. Est
Instrume=Newtonian
Aperture=.26
Fratio=6.
Power=63
Origin=Jet Propulsion Lab
Comment=Observing Site Unknown
Associated-File=AMATE001.FIT
Origin=JPL
```

To correctly process this update message, processing occurring during the *analysis phase* must be able to determine what type of observation was made and the identity of the observation target. The first piece of information is supplied by the attribute *Object* which identifies the observation target as P/CROMMELIN. Obviously, to correctly incorporate the object into the knowledge structure, the type of object which P/CROMMELIN represents must be determined. For example, it must first be determined that P/CROMMELIN is a comet, and that as a comet, it has certain other attributes (e.g., orbit, size, next predicted encounter, previous encounters, components).

The observation type is provided by the attribute *Dat-Type* and is identified as Visual Mag. Est. For the uninitiated user to make sense of this, the abbreviation must be expanded and its meaning understood. For example, because the observation involves measuring magnitude, the system should be able to reason what types of instruments might be used to make this type of observation. Such a capability aids additional processing of the message and aids the appropriate placement of the observation within the knowledge structure.

The *Instrume* attribute indicates that the measuring instrument is a Newtonian. The system will have to reason that a Newtonian instrument, which can be used to perform magnitude measurements, is a telescope, and that, as a telescope, it has certain characteristics. Within this context, processing of the update message can continue. Therefore, the attribute *Power*, when encountered, is now interpreted within the appropriate domain context (i.e., telescope), suggesting that *Power* indicates magnification used in the observation instead of an electrical power setting or other possible interpretation. Another example of inferring missing information from context can be seen when examining the attribute *File-Num*. This attribute provides a unique identifier but only within the organization responsible for the data. Meta-information inferred from *File-Num* must be determined from domain information contained within the KES and using various clues resident in the update message. From these clues it can be determined that *File-Num* refers to files maintained by the International Halley Watch.

However, this type of reasoning is not the only type required during the *analysis phase*. Different data formats use different attribute names to represent the same information or the same name to denote different information. Additionally, attribute formats indicating spatial and temporal information do vary, and the system must be able to understand how to interpret each type of information. This situation can be illustrated when comparing the update message 0001 with update message 0002.

Update Message 0002

```
Spacecraft_Name=Voyager_1
Mission_Phase=Jupiter_Encounter
Target_Body=Jupiter
Frame_Id=1309J1-059
Spacecraft_Clock_count=14641.14
Spacecraft_Event_Time=1979/01/06-
05:32:34
Earth_Received_Time=1979/01/07-00:20:51
Instrument_Name=Narrow_Angle_Camera
Instrument_Scan_Rate=1:1
Instrument_Shutter_Mode=NAONLY
Instrument_Gain_State=Low
Instrument_Edit_Mode=1:1
Instrument_Filter_Name=Orange
Instrument_Filter_Number=3
Instrument_Exposure_Duration=0.96000
Associated-File=C1464114.IMG
```

Notice the different names and formats for observation time, instrument used, and spatial specifications. In addition, notice how the attribute *Frame-Id* in update message 0003 uses a different format than update message 0002.

Update Message 0003

```
Spacecraft_Name=Voyager_1
Mission_Phase=Jupiter_Encounter
```

```
Target_Name=Jupiter_Magnetosphere
Frame_Id=16269.49
Frame_Period=48 <seconds>
Spacecraft_Clock_count=16269.49
Spacecraft_Event_Time=1979/060-12:24:36
Instrument_Name=Plasma_Wave_Spectrometer
Instrument_Mode=Waveform_Receiver
Instrument_Sampling_Rate=28800
Instrument_Lost_Samples=128
Associated-File=C1626949.IMG
```

It is possible that information used in the reasoning process during the *analysis phase* is inadequate to understand all pieces of the update message. When an update message is received in which some attribute or attribute value is unknown, further aid in interpreting the information will be required. In this case, the update message would likely be placed in a temporary buffer and a clarification request sent to the system operator. The update message is processed when the system operator responds to the clarification request. The operator's response might be as simple as identifying the unknown attribute in terms of an ISA-relationship with a known knowledge concept. For example, if reasoning during the *analysis phase* could not identify P/CROMMELIN, the system operator could inform the system that it is a comet and the update message could then be processed.[12] However, it is possible that a more complex alteration to the knowledge used by the reasoning system is needed. As a result, the expert system would have to be modified to reflect the new knowledge concept and its relationships to existing metainformation resulting in updates being processed as describe earlier.[13]

To better understand the nature of *analysis phase* processing and its relationship to automated updating of the knowledge structure, we have begun to closely examine what possible structure output of the *analysis phase* may take. For purposes of this discussion and to help the conceptual framework of analysis phase processing, initial results of our examination are presented here in terms of one of the example update messages provided earlier. Update message 0001 is reproduced below.

Update Message 0001

```
Object=P/CROMMELIN
File-Num=800200
Date-Obs=29/12/83
Time-Obs=.7600
Date-Rel=13/12/85
Discipln=Amateur
Long-Obs=999/99/99
Lat-Obs=+99/99/99
System=85000000
Spec-Evt=F
Dat-Type=Visual Mag. Est
Instrume=Newtonian
Aperture=.26
Fratio=6.
Power=63
Origin=Jet Propulsion Lab
Comment=Observing Site Unknown
Associated-File=AMATE001.FIT
Origin=JPL
```

The first step of the analysis process would examine the *Object* attribute, the first attribute provided in the update message, and would relate the attribute to the concept *target body*.[14] Establishing this relationship determines where in the knowledge structure *target body* is located. In this example, we assume that *target body* is directly related to the concept *Science Interests*.[15] Knowing this information results in the generation of the first segment of relevant concept/relationship information. Part of this information is given below:

```
Concept 1: Science Interests
Concept 1 Type: Structure
Concept 2: Target Body
Concept 2 Type: Structure
Relationship: Structure
Relevance View: Cometary Studies
```

This information identifies applicable concepts, their types, relationships existing among them, and associated relevance conditions. At this point in the process, only the appropriate relevance view is known. After all concept/relationship information is known, a comprehensive relevance condition would also be generated.

After the attribute has been processed, the attribute's value is processed. The system would then reason about P/CROMMELIN taking into account that it is a type of *target body*. The fact that P/CROMMELIN is a *comet* can then be determined.[16] Using the concept

[12] Such a scenario assumes the system operator is either knowledgeable concerning basic domain information or has access to reference material describing such information.

[13] This scenario might require contact with a domain expert to help define appropriate domain knowledge.

[14] The concept *target body* represents one node in the existing knowledge structure. In reality, the system would attempt to relate the attribute to as many concepts as possible.

[15] Keep in mind that the term 'concept' is used here to refer to any node in the knowledge structure. The term 'relationship' refers to nodal links.

[16] The actual relationship would be *ISA*.

target body as a starting point, the concept *comet* can then be located within the knowledge structure. In this case, *comet* has a direct relationship with *target body* and hence, the following concept/relationship information is generated.

Concept 1: Target Body
Concept 1 Type: Structure
Concept 2: Comet
Concept 2 Type: Value
Relationship: Value
Relevance View: Cometary Studies

Note that the relationship is now of the type *value* since *comet* is a valid value for the concept *target body.*

The next piece of information follows directly, linking the concepts *comet* and *P/CROMMELIN*.

Concept 1: Comet
Concept 1 Type: Value
Concept 2: P/CROMMELIN
Concept 2 Type: Value
Relationship: Value
Relevance View: Cometary Studies

Given this new concept/relationship structure, a user could indicate that the desired value for the concept *target body* could be either *comet* (all data sets pertaining to any comet are selected) or *P/CROMMELIN* (only those data sets dealing with this specific comet are selected). The user is allowed to specify the more general concept *comet* for the value of *target body* and then, if desired, return at a later time and define the concept more specifically by selecting a specific comet or group of comets of interest.

As with the concept *target body*, most concepts can only be assigned a single value.[17] However, there are concepts for which multiple values are acceptable. One of these concepts is *Instrume*. It is possible that some data sets can be generated by a collection of instruments working together.[18] Therefore, the concept *Instrume* would have to represented with multiple value capability in the output of the reasoning system .

In our example, when analysis processing arrives at the concept *Instrume*, it must be reasoned that this concept is the same as the concept *instrument* to be found in the knowledge structure. Like *target body*, *instrument* is related to the concept *Science Interests*. The following information would then be generated.

Concept 1: Science Interests
Concept 1 Type: Structure
Concept 2: Instrument
Concept 2 Type: Multiple Value Structure
Relationship: Structure
Relevance View: Cometary Studies

As before, the value of the *Instrume* attribute is then processed. The system determines that *Newtonian* is a type of the concept *telescope* which is a type of *instrument*. These concepts are located in the knowledge structure and the following information is generated.

Concept 1: Instrument
Concept 1 Type: Multiple Value Structure
Concept 2: Telescope
Concept 2 Type: Value
Relationship: Value
Relevance View: Cometary Studies

Concept 1: Telescope
Concept 1 Type: Value
Concept 2: Telescope Type
Concept 2 Type: Structure
Relationship: Qualification
Relevance View: Cometary Studies

Concept 1: Telescope Type
Concept 1 Type: Structure
Concept 2: Newtonian
Concept 2 Type: Value
Relationship: Value
Relevance View: Cometary Studies

Finally, there are times when concept defining information will need to be generated. Using update message 0001, one example is to define value units for the attribute *aperture*. *Aperture* must be processed in the context that it represents a qualification of the specified *instrument*. The reasoning system indicates that *aperture* specifies the aperture size of the *instrument telescope*. In addition, the reasoning system determines that defining information is required. As a result, the following information would be generated.

Concept 1: Telescope
Concept 1 Type: Value
Concept 2: Telescope Aperture
Concept 2 Type: Structure
Relationship: Qualification
Relevance View: Cometary Studies

Concept 1: Telescope Aperture
Concept 1 Type: Structure

[17] Within an AND condition, alternative values can be selected using an OR condition.

[18] A photometer attached to a telescope is one example.

Concept 2: .26
Concept 2 Type: Value
Relationship: value
Relevance View: Cometary Studies

Concept 1: Telescope Aperture
Concept 1 Type: Structure
Concept 2: Telescope Aperture Information
Concept 2 Type: Information
Relationship: information
Information: The size in meters of the effective aperture of the specified telescope.

When the user navigates to the value .26 for *aperture*, an *information* concept is provided which defines the value.

In the course of generating this type of output for each update message, it becomes clear that many redundant specifications exist. Such a situation does not pose a problem since the knowledge structure checks specified relationships and concepts, and if an exact match is found, the concept or relationship will not be redefined. It is important for two reasons that the complete set of concepts and relationships included in the update message be generated each time: the *analysis phase* does not maintain information about which relationships have already been defined, and users may not have the specified relationship within their views.

When all concept/relationship information is generated, the reasoning system then knows under what conditions each link is relevant. In this case, the following information has been understood:

Target Body - Comet or P/CROMMELIN
Instrument Type - Newtonian
Instrument Aperture - .26

This information would then have to be combined with other information to form the appropriate relevance condition for the link. As indicated earlier, a relevance condition can be more than a simple specification of attribute/value bindings. It can also contain complex logic based on other factors (see section 5.5).[19]

7.0 SUMMARY AND CONCLUSIONS

We have attempted in this paper to provided the underlying framework for what we consider to be two of the most crucial problems facing future, advanced information management systems: the design and implementation of an efficient knowledge structure supporting sophisticated user information access and manipulation, and the incorporation of automated knowledge structure update processing to provide an effective means for accomplishing knowledge structure evolution. As a result, the underlying knowledge structure for an earlier IIMS prototype has been extended and modified to handle not only semantic but also syntactic information, a capability made necessary by the requirement for the IIMS to be updated automatically. Our approach to knowledge structure development and modification has resulted in a significantly more flexible and easily modified knowledge structure that will be used in the future to continue to examine issues related to automated data and *metadata* updates for very large data/information systems.

REFERENCES

Carnahan, R.S., & Corey, S.M. (1989). Advanced information management and global decision making. *Proceedings of the Conference on Earth Observations and Global Change Decision Making: A National Partnership*, in press.

Carnahan, R. S., Corey, S. M., & Snow, J. B. (1989). A rapid prototyping/artificial intelligence approach to space station-era information management and access. *Telematics and Informatics, 6(3-4)*, 273-297.

[19] Relevance condition specification is beyond the scope of this paper.

ADAPTIVE PATTERN RECOGNITION BY MINI-MAX NEURAL NETWORKS
as a part of an intelligent processor

Harold H. Szu
NRL, Code 5756, Washington, D. C. 20375. Tel. (202) 767-1493

Abstract- In this decade and progressing into 21st Century, NASA will have missions including Space Station and the Earth related Planet Sciences. To support these missions, a high degree of sophistication in machine automation and an increasing amount of data processing throughput rate are necessary. Meeting these challenges requires intelligent machines, designed to support the necessary automations in a remote space and hazardous environment. There are two approaches to designing these intelligent machines. One of these is the knowledge-based expert system approach, namely AI. The other is a non-rule approach based on parallel and distributed computing for adaptive fault-tolerances, namely Neural or Natural Intelligence (NI). The union of AI and NI is the solution to the problem stated above.

The NI segment of this unit extracts features automatically by applying Cauchy simulated annealing [Phys. Lett. A122, p.157; Proc. IEEE, V.75,p.1538] to a mini-max cost energy function. The feature discovered by NI can then be passed to the AI system for future processing, and vice versa. This passing increases reliability, for AI can follow the NI formulated algorithm exactly, and can provide the context knowledge base as the constraints of neurocomputing. Such integration is exemplified by the pattern recognition Human Visual Systems; tracking of gray scaled objects for instance. Consequently, both AI and NI can work together to solve the same problem by unifying into an intelligent processor.

The mini-max cost function that solves the unknown feature can furthermore give us a top-down architectural design of neural networks by means of Taylor series expansion of the cost function. A typical mini-max cost function consists of (1) the sample variance of each class in the numerator, and (2) separation of the center of each class in the denominator. Thus, when the total cost energy is minimized, the conflicting goals of intraclass clustering and interclass segregation are achieved simultaneously. This Taylor expansion variable is a neuronic vector representation which traces along a Peano's curve. A selective space-filling capability exists when a more detailed spatial resolution becomes desirable at the picture where an interesting change occurs [IJCNN-90, D.C., p. II-76].

INTRODUCTION

Research and operations that support NASA's missions have experienced an increasing volume of data that requires automated information processing, among others (e.g. Discovery shuttle between the space station and the earth shown in Fig. 1 Top). One necessity is the next generation smart sensors. They are needed for two reasons. First, they are needed to perform multisensor data auto-fusion (between thematic mapper spectral band imageries and high spatial resolution imageries) in order to improve the picture resolution beyond the geometrical corrections and proper registrations. They are also needed to extract features to identify space rocket boosters shown in Fig. 1 center (provided by courtesy of T Dworetzky). From left to right, these are Goddard (1941), V-2 (German, 1944), Redstone (1961), Atlas Centaur (1962), Delta 3920 (1982), Titan 34D (1982), Saturn V (1967), Ariane (European 1979), Energia (Soviet 1987), and Conestoga II (Future). Automated feature extraction can also be useful to update maps as well as to help manage earth's resources. For example, an extra road through the palm forest was discovered by Environmental Research Institute Michigan (ERIM) in Fig. 1 (Bottom) D.

The trend in the modern telecommunications is toward multi-media, higher-speed and increased intelligence (Fig. 2 (a)). Thus, another application of intelligent machines is, according to NTT Review (Vol. 1, No.1 May 1989), a Broad-Band Integrated Service Digital Network (B-ISDN) that has been proposed and will probably undergo construction around 1995 (see Fig. 2 (b), used with permission). B-ISDN will have the capability of processing voice, images, and text, simultaneously based on neurocomputing.

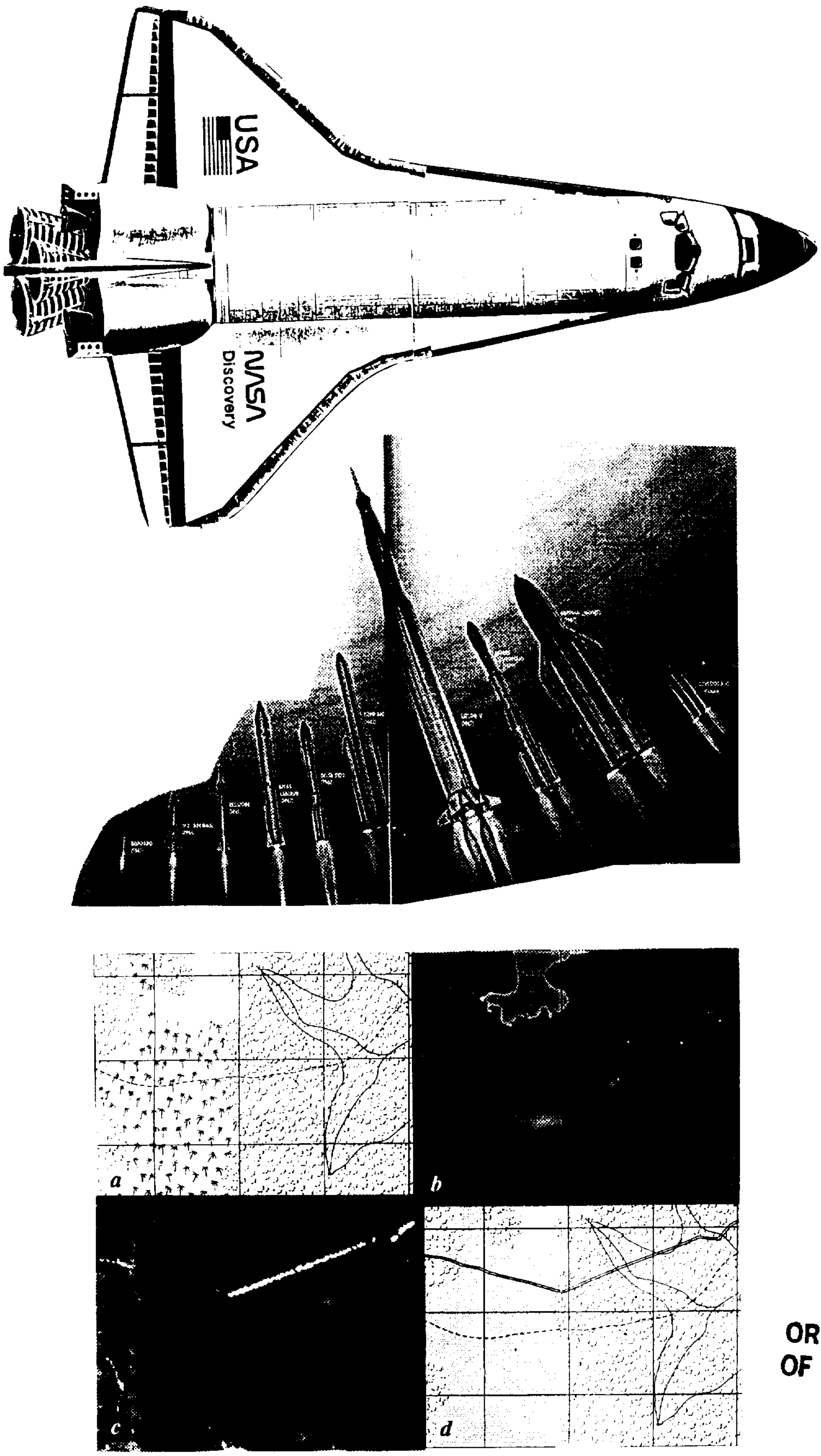


Figure 1. (Top) NASA's Space Shuttle Discovery. (Center) Feature extraction to identify various space rocket boosters. (Bottom) Automatic feature extraction to update maps and to help manage earth's resources.

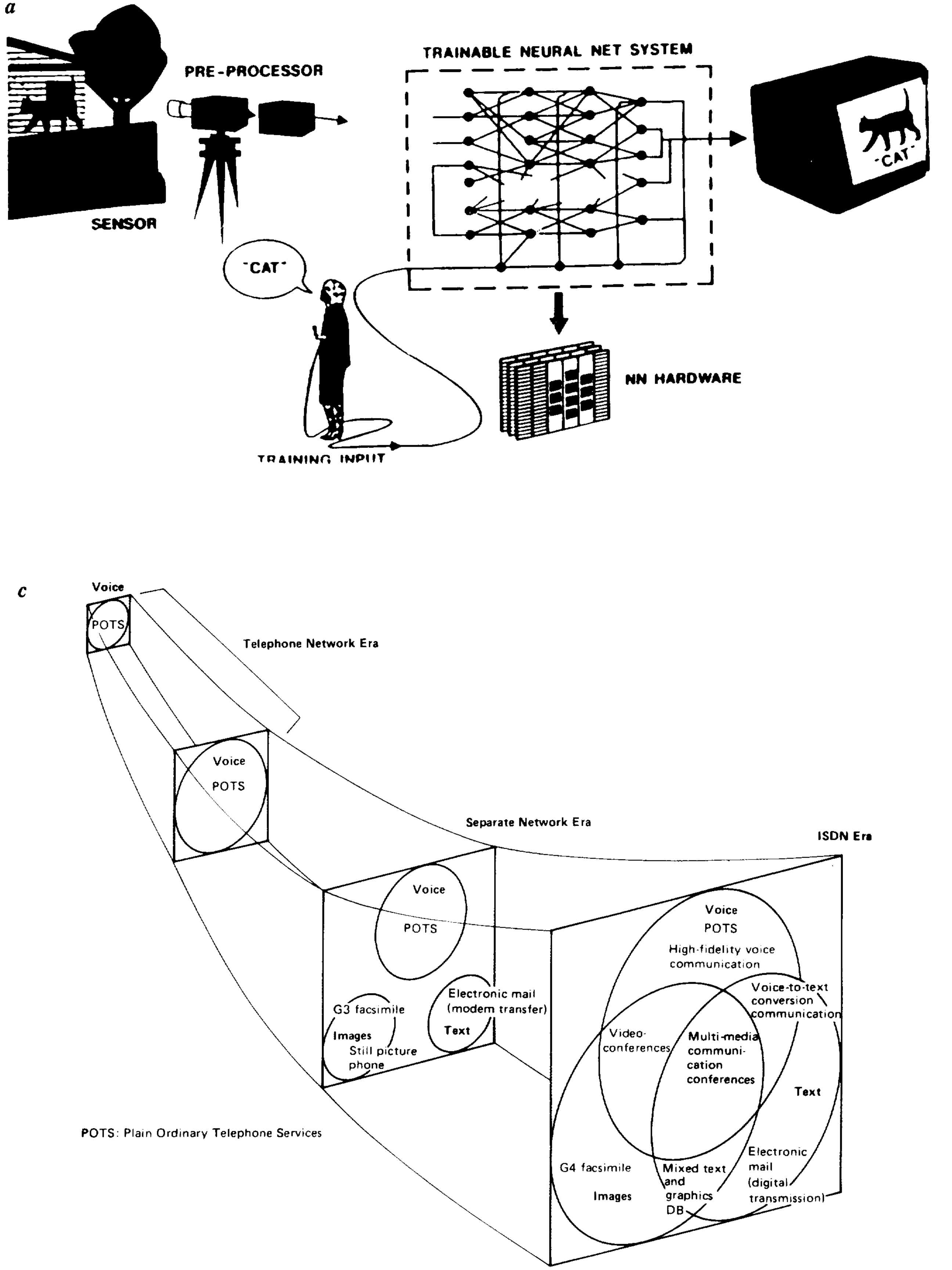

a
SENSOR
PRE-PROCESSOR
TRAINABLE NEURAL NET SYSTEM
"CAT"
"CAT"
NN HARDWARE
TRAINING INPUT
c
Voice
POTS
Telephone Network Era
Voice
POTS
Separate Network Era
Voice
POTS
G3 facsimile
Images
Still picture phone
Electronic mail (modem transfer)
Text
ISDN Era
Voice
POTS
High-fidelity voice communication
Voice-to-text conversion communication
Video-conferences
Multi-media communication conferences
Text
G4 facsimile
Images
Mixed text and graphics DB
Electronic mail (digital transmission)
POTS: Plain Ordinary Telephone Services

Figure 2. (a) The trend towards Multi-media Communication. (c) Mixing of voice and image spectral components by an associative memory.

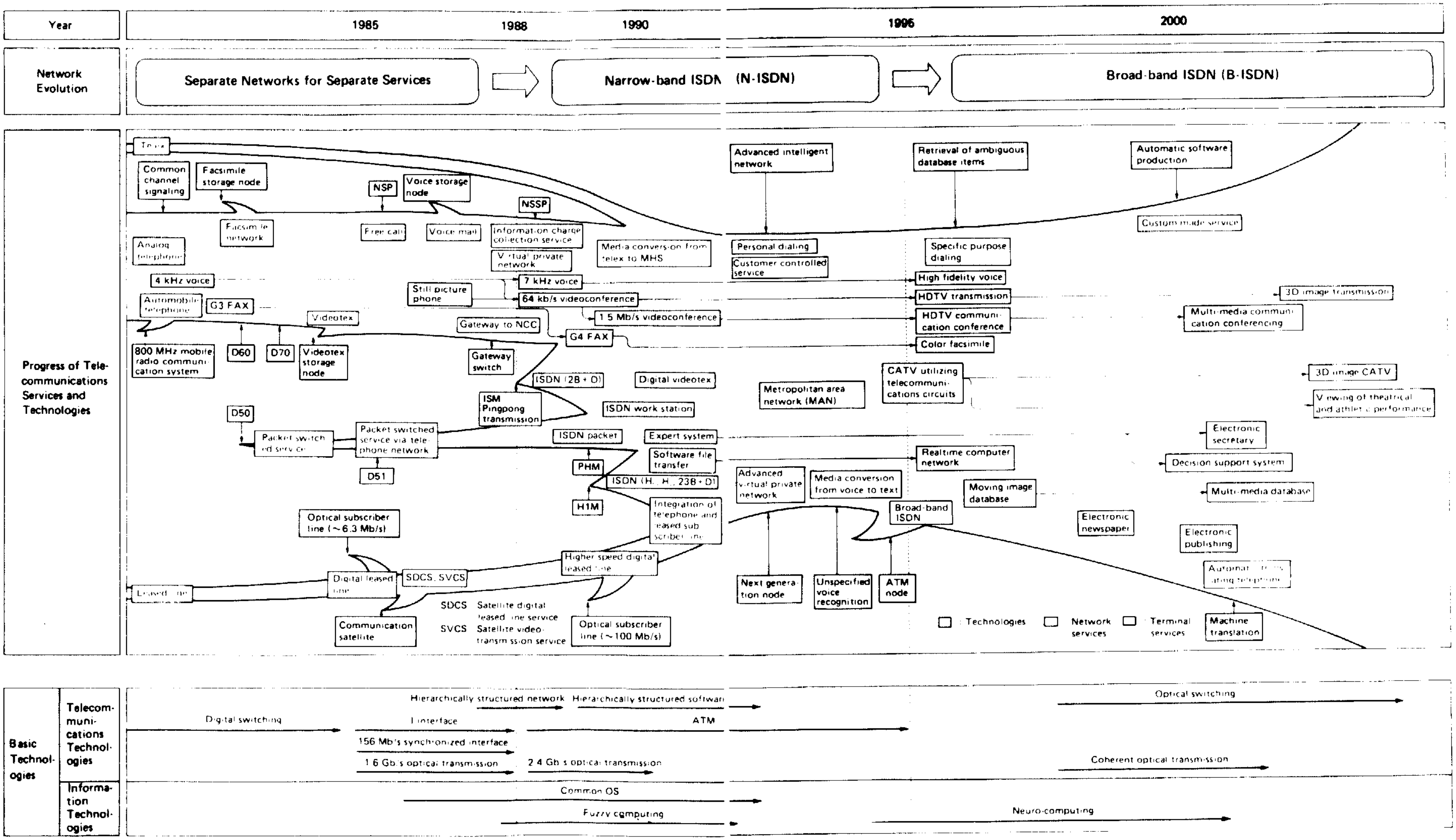

Year
1985
1988
1990
1995
2000
Network Evolution
Separate Networks for Separate Services
Narrow-band ISDN (N-ISDN)
Broad-band ISDN (B-ISDN)
Progress of Telecommunications Services and Technologies
Telex
Common channel signaling
Facsimile storage node
Facsimile network
NSP
Voice storage node
NSSP
Free call
Voice mail
Information charge collection service
Virtual private network
Media conversion from telex to MHS
Analog telephone
4 kHz voice
Automobile telephone
G3 FAX
Videotex
Still picture phone
7 kHz voice
64 kb/s videoconference
1 5 Mb/s videoconference
Gateway to NCC
G4 FAX
800 MHz mobile radio communication system
D60
D70
Videotex storage node
Gateway switch
ISDN (2B + D)
Digital videotex
ISM Pingpong transmission
ISDN work station
D50
Packet switched service
Packet switched service via telephone network
D51
ISDN packet
Expert system
Software file transfer
PHM
ISDN (H, H, 23B + D)
H1M
Integration of telephone and leased subscriber line
Optical subscriber line (~6.3 Mb/s)
Higher speed digital leased line
Digital leased line
SDCS, SVCS
Leased line
Communication satellite
SDCS Satellite digital leased line service
SVCS Satellite video transmission service
Optical subscriber line (~100 Mb/s)
Advanced intelligent network
Personal dialing
Customer controlled service
Retrieval of ambiguous database items
Specific purpose dialing
High fidelity voice
HDTV transmission
HDTV communication conference
Color facsimile
Metropolitan area network (MAN)
CATV utilizing telecommunications circuits
Advanced virtual private network
Media conversion from voice to text
Realtime computer network
Moving image database
Broad-band ISDN
Next generation node
Unspecified voice recognition
ATM node
Automatic software production
Custom made service
3D image transmission
Multi-media communication conferencing
3D image CATV
Viewing of theatrical and athletic performance
Electronic secretary
Decision support system
Multi-media database
Electronic newspaper
Electronic publishing
Automatic translating telephone
Machine translation
□ Technologies
□ Network services
□ Terminal services
Basic Technologies
Telecommunications Technologies
Information Technologies
Digital switching
Hierarchically structured network
Hierarchically structured software
I interface
ATM
156 Mb/s synchronized interface
1 6 Gb/s optical transmission
2 4 Gb/s optical transmission
Optical switching
Coherent optical transmission
Common OS
Fuzzy computing
Neuro-computing

Figure 2. (b) The NTT Review forecasts a Broad-Band Integrated Service Digital Network (B-ISDN) by 1995 using neurocomputing.

Associative memory can mix the voice and image spectral components by vector outer products (shown in DARPA's Report as a dotted matrix array in Fig. 2 (c)). This cross correlation information can be processed at the data rate of about 3 Gb/s. Usually such a high data rate requires optical computing based on optical switching and coherent optical transmissions. However, neurocomputing's debut in telecommunication is predicted by NTT to be five years earlier than optical computing, despite extensive research efforts in optical computing by AT&T & others.

REVIEW OF NEURAL NETWORK LEARNING ALGEBRA

Neural network computing is a nonlinear system that satisfies 4 none-principles with the fifth none-principle remains to be worked out. These are: (1) a none-linear threshold logic of neurons, (2) a none-local associative memory, (3) a none-stationary neurodynamics, and (4) a none-convex system energy, meaning more than one extremum in the energy landscape. The first one is known to us 30 years ago, when the Rosenblatt's perceptron was proposed to be a random collection of neurons. It had been shown by Minsky and Papert to be insufficient for the natural intelligence, and thus giving the need to the birth to AI. These 4 none-principle can be approximated by (1) piecewise-linear, namely binary neurons, (2) piecewise-local, namely the rank-1 vector outer product, (3) piecewise stationary, namely iterative revisions, and (4) piecewise convex, namely local gradient descents. In these controlled approximations, these interwoven complex principles become decoupled and amenable to powerful computer simulations. Since then NI has been coming a long way, there remains a missing fifth none-principle. Such a none-programming learning principle has been claimed by some, but the hidden teachers/programmers remain to be unraveled to most of us for pedagogical reasons. This is the state of the art of neurocomputing theory.

Neurocomputing learning algebra are based on the variants of Hebbian ideas. Giving two random inputs of two neuron firing rates about 100 Hz, u_i u_j, there are limited algebraic structures that one can manipulate with to extract meaningful information. If the change of synaptic weight at the ith and the jth interconnection, ΔW_{ij}, should be related to the inputs as follows:

- **Correlation Learning:** $\Delta W_{ij} \approx u_i \, u_j$

  (maximum information-exchange rule between a pair of random firing rates)
- **Gradient Learning :** $\Delta W_{ij} \approx (D_i - v_i)\, u_j$

  (Error correction by a pre-set output goal D_i that decides when the change of actual output v_i stops: the delta rule)
- **Competitive Learning** $\Delta W_{ij} \approx u_i\,(u_j - W_{ij})$

  (any change must balance against the old cluster establishment w_{ij})
- **Differential Learning** $\Delta W_{ij} \approx (du_i/dt)\,(du_j/dt)$

  (Only time rate changes, derived by Taylor series expansion of $u_i(t)$, matters)

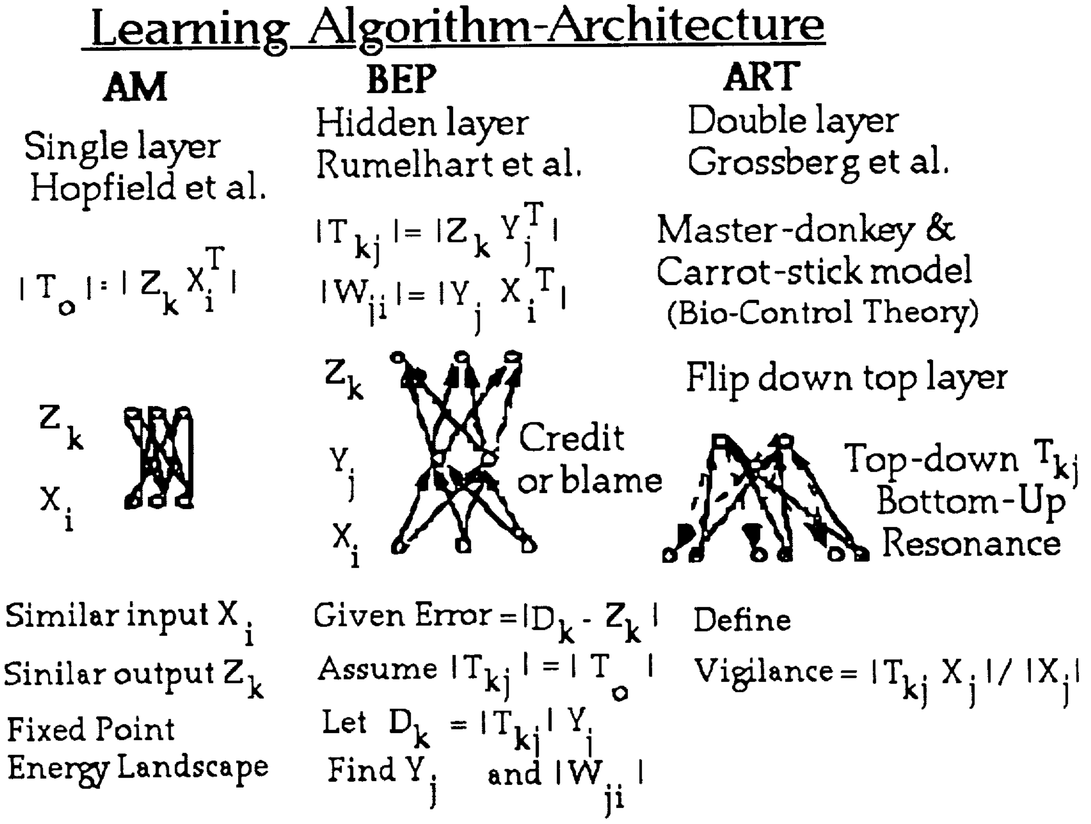
REVIEW OF NEURAL NETWORK ARCHITECTURES

Neural network architectures are important for parallel and distributed computing. There are: one layer of Hopfield's Associative Memory (AM), two layers of Grossberg's Adaptive Resonance Theory (ART), and three layers of Rumelhart's Back Error Propagation (BEP), as shown schematically in Fig. 3, Learning Algorithm-Architecture as follows.

- In the left hand column of Fig. 3, similar inputs X_i are mapped into similar outputs Z_k in a feature space. Such a (hetero-associative) matrix memory is formed by the vector outer product forming a matrix denoted as $|Z_k X_i^T|$, where the superscript T stands for the transpose of the column vector X (indexed with the component i) and the column vector becomes a row vector. Matrix memory is a static version of Hopfield neural networks, because of the fixed point coding between the input and the output requires no learning. By a fixed point coding we mean that "write-by-outer-product" and "read-by-inner-product" and using the matrix-vector operation without iterations.
- In the middle column of Fig. 3, when the similar inputs produce the surprising outputs, an extra layer is introduced to interpolate these abnormal results by means of supervised training. The difference $|D - Z|$ from the output Z with respect to the desired output D is considered to be the error propagates backward by means of a

local gradient descent methodology. The system can have the potential for the generalization. There are several theories about the size of the so-called hidden layer and the ability to do the abstraction (with more neurons than that of input nodes) or the generalization (with a fewer neurons). The degree of freedom must match the number of sample classes to be classified based on the orthogonal feature space min-max concept described in the following section. In such a quasi-orthogonal storage, this rule seems to be reasonable in assigning credit-or-blame.

- When the desired output D is not yet known, Grossberg model of Adaptive Resonance Theory (ART) becomes handy. It might be thought as to flip down the unknown output layer in order to compare the unknown input directly as shown in the righthand column of Fig. 3. The master has its own top-down wires T_{jk} (shown by dotted lines), while the donkey has its own bottom-up wires b_{ij}. In order to carry out automatically the clustering technique by following the leader, the top layer master puts his feet into donkey's input x_j to test his own normalized prediction $|S <x_j | T_{jk} | x_k> | / | S <x_j | x_k> |$ with a predetermined parameter, called the vigilance parameter between 0.5 to 0.9. Therefore, the difference between the traditional control theory with the negative feedback and the neural network is that both the incentive/carrot and the punishment/stick are used in the biological model having both the excitation and the inhibition exerted at different parts of the self-organized system.



Learning Algorithm-Architecture

AM	BEP	ART
Single layer Hopfield et al.	Hidden layer Rumelhart et al.	Double layer Grossberg et al.
$\lvert T_o \rvert = \lvert Z_k X_i^T \rvert$	$\lvert T_{kj} \rvert = \lvert Z_k Y_j^T \rvert$ $\lvert W_{ji} \rvert = \lvert Y_j X_i^T \rvert$	Master-donkey & Carrot-stick model (Bio-Control Theory)
		Flip down top layer
Similar input X_i Sinilar output Z_k Fixed Point Energy Landscape	Given Error $= \lvert D_k - Z_k \rvert$ Assume $\lvert T_{kj} \rvert = \lvert T_o \rvert$ Let $D_k = \lvert T_{kj} \rvert Y_j$ Find Y_j and $\lvert W_{ji} \rvert$	Define Vigilance $= \lvert T_{kj} X_j \rvert / \lvert X_j \rvert$

Figure 3. Review of Learning Algorithm-Architecture.

An interesting taxonomy dilemma about counting of layers is due to the ambiguity of counting about layers of neurons or about layers of interconnects. The single-layer Hopfield architecture seems to have two layers of neurons, with respect to the three layers of Rumelhart architecture. On the other hand, the Hopfield architecture is considered to be a single layer on a VLSI design. This dilemma may be resolved by asking: What is more important in counting, the layer of interconnect synaptic weights, or the layer of neurons ? Since the synaptic weights contain the important memory information, then Hopfield's network should be counted as one layer.

Designs of Energy Cost Functions in A Neuronic Vectorial Representation

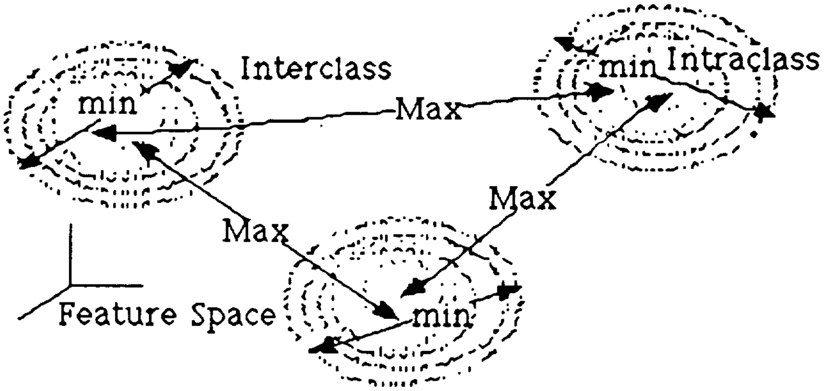
An important question for practical applications is how to speed up the training process and to insure a fast convergence of weight adjustment? We have suggested a general procedure of Taylor series expansion of the clustering-declustering mini-max energy to estimate the synaptic weights. Here, we extend the procedure by a self-consistently variational technique to make the truncated higher order terms of the Taylor series negligible.

A top-town design of a hard-wired neural network algorithm has been initiated by Hopfield, et al, for constrained optimizations. We consider a supervised top-down design goes beyond Hopfield's attempt. The minimum clustering of the alike and the maximum declustering of the disalike seems to be two contradicting goals. A tradeoff can be mathematically constructed by the linear combination of those pairs alike in the numerator and the pairs of disalikes in the denominator of a mini-max energy formalism (schematically shown in the cost energy expression of Fig.4).

Top Down Design of Hard-Wired Neural Networks

Mini - Max Energy Principle

$$\text{Energy} = \sum \frac{1}{|\text{Interclass Distance}|} + \sum |\text{Intraclass Distance}|$$



Taylor Expansion to derive multiple layer interconnects

$$\text{Energy} = -\frac{1}{2}\sum_{i,j} T_{i,j} V_i V_j - \frac{1}{6}\sum_{i,j,k} T_{i,j,k} V_i V_j V_k - \cdots$$

Figure 4. A top-down design of Neural Networks.

Let us consider some application of pattern classifications. The class of physically different objects {o, O, p, P, q, Q} need to be cleverly pre-processing by a smart sensor mimicking our eyes or by ourselves and then endow our wisdom about how we classify the set with a functional mapping into a feature space { o(V_i), O(V_i), p(V_i), P(V_i), q(V_i), Q(V_i)} spanned by a sufficient set of neurons V_i mimicking the human visual system of the brain (Szu & Scheff 1989). The first term of the energy in the denominator is similar to the Coulomb energy of repulsive electric charges (reduced Coulomb energy model of Cooper, et al., and Lorentz forces of Sayeh, et al), and the second term in the numerator is similar to the least square method(when in an arbitrary power becomes Kohonen's kth norm clustering method).

While the first order derivative is reserved for the aforementioned neurodynamics equations, the second order derivative when it is evaluated at the equilibrium value: $V_i=V^{(o)}_{i,}$ $V_j=V^{(o)}_j$ becomes the Taylor's coefficient

$$T^{(o)}_{i,j} = (\partial^2E/\partial V_i\partial V_j)\,|_{Vi=V^{(o)}i,Vj=V^{(o)}j}$$

Then, the Hopfield-like hard-wired interconnect $T^{(o)}_{i,j}$ become soft-wired $T_{i,j}$ by means of the Hebbian learning that make the cubic order negligible.

$$T_{i,j} =T^{(o)}_{i,j} + \varepsilon\, \delta V_i\, \delta V_j$$

$$T_{i,j,k}\,|_{Vi=V^{(o)}i+\delta Vi,Vj=V^{(o)}j+\delta Vj} << T^{(o)}_{i,j}$$

Similarly, the procedure can be analogously extended to the three layers:

$$T_{i,j,k}=T^{(o)}_{i,j,k} + \varepsilon\, f(\delta\, V_i\delta V_j\delta V_k)$$

which makes the next fourth order derivative negligible. The case of hidden layer architecture means that $T_{i,j,k}$ is a block-diagonalized tensor of which the input ith layer can not communicate with the kth layer output layer without going through the jth hidden layer of neurons.

We can show that a single layer of a fully interconnected Hopfield network of five neurons of 25 interconnects can be reduced by the use-it-or-lose-it principle to 6 interconnects. Without actual physical rearrangement, it becomes topologically equivalent to a three layer of Minsky nets by clamping 2 input neurons and 1 output neuron to be trained repeatedly with the "exclusive OR" input-output relationship. This illustrates the second computing principle that can not only be used to determine the learning algorithm but also used to derive the neural network architectural change consistently.

Experimental aspect of the unified learning theory has been demonstrated by NTT scientists using several life neurons, extracted from the hippocampus of chicken brain. In delayed video recording they have shown that neuronic hair fibers T_{ij} grow for seeking out the nutrition and other neurons, in a competitive learning fashion. The winning hair fiber has grown fatter into a mature axonic interconnect, while the other loser shrinks off, on an electronic chip substrate covered with the life sustaining liquid. The present unified theory is possible to explain such a growing synapse because of the extended McCulloch-Pitts neuron model with two transfer functions for two independent degrees of freedom, namely the sigmoidal firing rate transfer function and the synaptical transfer function. Such a model has been coined with a name of the hairy neuron neural networks (Szu, 1989).

NEUROCOMPUTING IS MORE THAN PARALLEL COMPUTING

The famous von Neumann bottleneck, 10^9 operations per second (ops), for a sequential computer has been circumvented by parallel computing models which require lock steps among multiple processors controlled by a precision clock cycle that has unfortunately created the second bottleneck, 10^{12} ops, (that I wish to call) **the five W bottleneck**, namely "who should do what, when, where and how" bottleneck, due to the necessary trade off between the actual execution and the communication for timing and assigning jobs among multiple pipe lines. Therefore, the following asynchronous neurocomputers are fundamentally important and can make possible a cheaper VLSI fabrication of neurocomputers. Although the fabrication advantages without the demand of timing accuracy is conceivable, but without neuronic processor timing the dynamics about when and how the collective computing is finished requires mathematical insurance. Thus, we will prove three theorems for three neurocomputing learning mechanisms with hard-wired, soft-wired, and brittle-wired interconnects. Our purpose is to point out the possibility of allowing the system to determine its own topological structure, by means of a dynamically reconfigurable hairy neuron model described below. In order to minimize the overall energy, dynamically reconnected neurofilaments T_{ij} (located at the protein-mediated output axons) can play an equally important role as the synaptic junction W_{ik} weight adjustments (located at the ion-mediated input dendrite tree). The extra degree of freedom of the hairy neurons is the synaptic transfer function having a nonnegative slope

$$T_{ij}=f(W_{ik}); \qquad (dT_{ij}/dW_{ik}) \geq 0$$

while the McCulloch-Pitts neuron model has one internal degree of freedom prescribing the firing rate transfer squash function

$V_i = g(U_i)$; $(dV_i/dU_i) \geq 0$

The following three convergence theorems all depend on the mathematical truth that (d (any real quantity)/d $t_i)^2 \geq 0$ with respect to any time axis:

$t_i = t_i^{(o)} + \varepsilon_i t$,

where the information arrival time has an arbitrary initial time $t_i^{(o)}$ and a positive time scale factor $\varepsilon_i > 0$ with respect to a collective or universal time axis t.

(1) Hopfield-like Asynchronous Computing by Hard-Wired Nets $E_1(V)$

We consider first a system of a hard-wired neural networks. We assume a network activity energy $E_1(\mathbf{V})$ in terms of the output firing rate vector **V** with the components V_i whose i index runs from one neuron to a million, e.g. the mega-Cray. We can use either $E_1(V_i)$ or $E_1(\mathbf{V})$. The input firing rate to the ith neuron is wired according to the McCulloch-Pitts model with the bias Θ_i:

$$U_i = \Sigma_j W_{ij} V_j + \Theta_i. \qquad (1)$$

The synaptic weight W_{ij} at the jth junction of the ith neuron input dendrite has a physical gap, analogous to the spark plug, through which the ion-mediated firing rates from other outputs V_j are collected. Then, Hebbian learning would mean the changing of the spark plug gap for tuning up the car engine firing rates. Due to the diffusion of discrete ions through those synaptic junctions, the firing rate fluctuates like a discrete time series at the molecular time scale t in the order of one millisecond. The information flow with a reduced fluctuation of the neurotransmitters plays an important annealing role for the global convergence of the neurodynamics.

Each neuron can be operated at its own time axis:

$$t_i = t_i^{(o)} + \varepsilon_i t, \qquad (2)$$

where the information arrival time has an arbitrary initial time $t_i^{(o)}$ and a positive time scale factor $\varepsilon_i > 0$ with respect to a collective or universal time axis t. This asynchronicity is essential to account for different information flow rates due to the biological inhomogeneity at neuronic level.

The total input is instantaneously mapped to the output by a nonlinear transfer function g,

$$V_i = g(U_i) \qquad (3a)$$

A squash function known in biology as a sigmoidal function is often used

$$g(x)=1/(1 + \exp(-x)) \qquad (3b)$$

for the simplicity of the analytic slope:

$$dg/dx = g(1-g) \geq 0, \qquad (3c)$$

which vanishes at g=0 when the neuronic decision means no, or at g=1 meaning yes. This set of Eq. (3a, b, c) describes an analog model of McCulloch-Pitts neurons. The original proof of convergence by Hopfield uses explicitly a quadratic energy expression among neurons for easy analog VLSI implementation. An independent proof has been given by Cohen and Grossberg that does not require the symmetry property of interconnects.

Each fine grained processor has been modeled in this paper by a different propagation speed governed by the first order equation:

$$(dU_i/dt_i) = -(\partial E_1(\mathbf{V})/\partial V_i), \qquad (4)$$

driven by a local energy gradient.

The collective answer should emerge at (dE/dt)=0 when the seemingly random computing without the lock-clock synchronizations. With respect to the collective time, the following macroscopically irreversibility: $(dE/dt) \leq 0$ will be guaranteed.

Theorem I: Asynchronous Convergence based on $(dE_1(V)/dt) \leq 0$.

If the neural network energy $E_1(\mathbf{V})$ depends only on the set **V** of all output firing rates V_i, and if and only if an arbitrary transfer function, $V_i = g(U_i)$ has a non-negative slope: $(dV_i/dU_i) \geq 0$, then the change of each neuron input U_i governed by its own time axis, through the first order dynamics: $(dU_i/dt_i) = -(\partial E_1(\mathbf{V})/\partial V_i)$ where $t_i = t_i^{(o)} + \varepsilon_i t$ with $\varepsilon_i > 0$, will guarantee the monotonic convergence $(dE_1/dt) \leq 0$.

Proof: The differential increment of in time must maintain the direction of the time flow, Eq. (3b) implying a positive characteristic factor,

$dt_i = \varepsilon_i\, dt$,or, $(dt_i/dt)= \varepsilon_i > 0$

The energy-gradient is so-to-speak the force upon the axonic output that changes the firing rate of the total dendritic input Eq. (1). Nonetheless, the global energy converges with respect to the collective or universal time t.

$$(dE_1(V_i)/dt) = \Sigma_i\, (\partial E_1/\partial V_i)\, (dV_i/dt_i)(dt_i/dt) \qquad (5a)$$

$$= -\Sigma_i\, \varepsilon_i\, (dU_i/dt_i)\ \ (dV_i/dt_i) \qquad (5b)$$

$$= -\Sigma_\iota\, \varepsilon_i\, (dU_i/dt_i)^2\, (dV_i/dU_i) \qquad (5c)$$

$$\leq 0\,. \qquad (5d)$$

Eq. (5a) is obtained by the chain rule of differentiation; in Eq. (5b), use is made of Newtonian Eq. (4) to eliminate the the energy slope ; Eq. (5c) is merely the identity $(dV_i/dt_i)=(dV_i/dU_i)(dU_i/dt_i)$ used to produce the second power of (dU_i/dt_i) in Eq. (5c). The last inequality Eq. (5d) is based on the mathematical truth that the square of arbitrary real number

$$(dU_i/dt_i)^2= (\text{Real Numbers})^2 \geq 0$$

must be nonnegative in any time scale.

In the general convergence proof for arbitrary time axis t_i with $\varepsilon_i > 0$, we require no detail structure of the energy function, other than once differentiable. Thus, we have indeed verified the intuition that nothing changes $(dE_1/dt) = 0$ at the moment of convergence. This theorem may be called the first asynchronous neurocomputing principle that predicts the macroscopic irreversibility $(dE_1/dt) \leq 0$ from the microscopic reversible but time-asynchronous neurodynamics Eq. (4). The irreversibility is due to the necessary and sufficient condition Eq. (2) of the nonlinear transfer function g (that is equivalent to the stosszahl Ansatz of the binary collision transfer function in the Boltzmann Transport Equation). Although the proof similar to the Lyaponov theorem in the standard control theory, the learning mechanism in bio-control theory has been left unanswered.

(2) Rumelhart-like Weight-Adjustment Learning: Soft-Wired $E_2(W_{ij})$

Due to the biological inhomogeneity, the energy gradient descent methodology may be slightly generalized to a time-asynchronous learning algorithm that each neuron could have its own time axis

$$(dW_{ij}/dt_i) = -\,(\partial E_2(W_{ij})/\partial W_{ij}), \qquad (6a)$$

$$dt_i = \varepsilon_i\, dt\,, \qquad (6b)$$

Rumelhart, et al., has applied Eq. (6) to a feed forward and fixed layer architecture, within the synchronized layer of neurons: $\varepsilon_i= 1$. A slightly generalized convergence proof of time-asynchronous neurocomputing is given as follows:

Theorem II: Synaptic Adjustment Convergence:

$$(dE_2(W_{ij})/dt) = \Sigma_i\, (\partial E_2/\partial W_{ij})\, (dW_{ij}/dt) \qquad (7a)$$

$$= -\Sigma_i\, (dW_{ij}/dt_i)\, (dW_{ij}/dt_i)(dt_i/dt) \qquad (7b)$$

$$= -\Sigma_i\, \varepsilon_i\, (dW_{ij}/dt_i)^2 \qquad (7c)$$

$$\leq 0 \qquad (7d)$$

The adjustment of the synaptic weights W_{ij} can be derived implicitly in terms of the square error of the desired output D from the actual output V, when a given input U is fed into the layered network. Such a methodology is known as the **backward-error-propagation** resulting in a delta learning rule to assign the credit or the blame to other layered neurons behind them. To illustrate both energy functions $E_1(V_i)$ and $E_2(W_{ij})$, we assume

$$E_2(V_i(W_{ij})) = (1/2)\ \Sigma_i\, (\, D_i\ -\ V_i)^{\,2} \qquad (8)$$

to be the square error of the desired response D_i from the actual output V_i, which, in terms of the analytical transfer function g of the input $U_i=\Sigma_j W_{ij}\, V'_j\ + \Theta_i$ Eq. (1), are the upward link synaptic weights. We denote the set of (input, actual output, desired output) respectively as $(U_i\,, V_i\,, D_i)$. If there is no error: $(V_i\ -\ D_i) = 0$, no learning takes place. The upward link weights W_{ij} are adjusted to reduce the difference, by multiplying the time-dependent factor ε_i to both hand sides of Eq. (6a).

$$\varepsilon_i(dW_{ij}/dt_i) \equiv \Delta W_{ij} = -\varepsilon_i(\partial E/\partial W_{ij})$$
$$= -\varepsilon_i\{(\partial E/\partial V_i)(dV_i/dU_i)\}(\partial U_i/\partial W_{ij})$$
$$= -\varepsilon_i\{(V_i - D_i)V_i(1 - V_i)\} V'_i \qquad (9a)$$

where the straightforward differentiation has produced the result.

The delta learning formula is the input energy change: $-\{(\partial E/\partial V_i)(dV_i/dU_i)\} = -(\partial E/\partial U_i) \equiv \delta_i$ with respect to the top layer input: $U_i=\Sigma_j W_{ij} V'_j + \Theta_i$ in terms of the upward synaptic links W_{ij}. Such an energy change at the top layer input is propagated downward to the the input energy change with respect to the hidden layer input: $U'_k = \Sigma_m W'_{km} V''_m + \Theta'_k$, in terms of the downward synaptic links W'_{kj}

$$\delta_i \equiv -(\partial E/\partial U_i) = -\Sigma_k (\partial E/\partial U'_k)(\partial U'_k/\partial U_i)$$
$$= \Sigma_k \delta'_k \Sigma_m(\partial U'_k/\partial V''_m)(\partial V''_m/\partial V_i)(dV_i/dU_i)$$
$$\cong (dV_i/dU_i) \Sigma_k \delta'_k W'_{ki} \qquad (9b)$$

where the approximation equality sign $\cong$ is due to the replacement of the unknown top layer input V_i with the known bottom layer input V''_m. Thus, the delta learning rule remains to be approximately independent of neuronic time axes.

$$\delta_j = V_j(1 - V_j)\Sigma_k \delta'_k W'_{kj} \qquad (9c)$$

(3) Morphology Convergence for Hairy Neurons with Brittle-wired $E3(V_i;T_{ij})$

In this section, we wish to formulate a set of neurodynamics equations which can settle itself into an appropriate network architecture, e.g. one layer of Hopfield, three layers of Rumelhart, or two layers of Grossberg. Neurophysiological experiments have recently shown that an active neuron can grow hairy neurofilaments, denoted as T_{ij}, in competing for nutritions and networking partnership against other neurons, and has been called a hairy neuron model(Szu 1989). The distinction between input synaptic weight W_{ik} from the output axonic neurofilament T_{ij} is necessary because of the recent neurophysiological experiments: (1) the use-it or lose-it synaptic pruning in one eye jack of a new born kitten, and (2) the actin protein generating the growth of neurofilaments. These neurofilament hair lines are competing for food and partnership. The winner grows fatter, while the loser shrinks thinner. The active growth of neurofilament T_{ij} reaches out and touches other neuron, and becomes eventually matured and retracts itself in forming a physical gap, the synaptic junction W_{ik}, for better resistive control of the ion diffusion potential without the initial direct contact. In order to take into account the possibility of the pruning of synapses W_{ik} (1), and the active growth of neurofilaments T_{ij} (2), the synaptic weights W_{ik} at the ith neuronic dendrite tree and kth junctions are assumed to be dormant variables, while the neurofilaments T_{ij} located at the ith axonic output can grow into the jth neuron with the active tread-mill microtube assembly mechanism. Thus, we have extended the classical McCulloch-Pitts neuron model, Eq. (2), to include one more degree of freedom, such as the synaptic transfer function

$$T_{ij}=f(W_{ij}) \qquad (10a)$$

between the axonic filaments T_{ij} (protein actin-driven for dynamic growing/pruning) and the dendrite synapses W_{ij} (positive ion-driven firing rates). The biological survival principle, use it or lose it, can be applied to the neuron level to explain the observed fact of a reduced synaptic gap density by a pruning mechanism in the one eye jack experiment on a new born kitten. In this experiment, a patch was place over the eye of a new born kitten. The post-natal development of its brain had no optical inputs and the optical processing neural networks died off leaving the kitten normal eye function blind. It will take a life long training to regain the binocular vision. The synaptic transfer function Eq. (7a) becomes, in the new born or high gain limit, a binary step function of the threshold b and the step size a.

$$f(x) = a\ \mathrm{step}(x-b) \qquad (10b)$$

which shows the absorption of synapses below the threshold utility frequency b. In general, the synaptic transfer function has a non-negative slope

$$(dT_{ij}/dW_{ij}) \geq 0. \qquad (10c)$$

Both nonlinear transfer functions, Eqs(3,10), assume the input firing rate U_i and the input dendrite weight W_{ij} to be dependent variables. The energy $E_3(V_i;T_{ij})$ unified both Hopfield-like $E_1(V_i)$ and Rumelhart-like $E_2(W_{ij})$ is proposed for morphological reconfiguration of neural networks. The total system energy $E(V_i;T_{ij})$ is an analytic function of two independent dynamic variables representing neuronic outputs: the fast output firing rates V_i and the slow axonic pairing rate T_{ij} between the ith and the jth neurons. The local gradient descent learning algorithm of Rumelhart et al. is slightly generalized

$$(dW_{ij}/dt_i) = - (\partial E_3/\partial T_{ij}), \qquad (11)$$

which in a fixed and feed forward layer architecture is reduced to the back-error-propagation model of Rumelhart et al. if the synaptic transfer function becomes an identical mapping, meaning $T_{ij} = W_{ij}$ and the learning happened at one time scale $t_i=\varepsilon_i t$ with $\varepsilon_i=1$. The following proof of a global convergence of such a sophisticated learning with adaptive morphology is mathematically isomorphic to the case of fast time scale:

$$(dE_3(V_i;T_{ij})/dt = \{\Sigma_i(\partial E_3/\partial V_i)(dV_i/dt_i)+\Sigma_{ij}(\partial E_3/\partial T_{ij})(dT_{ij}/dt_i)\}(dt_i/dt) \qquad (12a)$$

$$= - \Sigma_i\varepsilon_i(dU_i/dt_i)\ (dV_i/dt_i)- \Sigma_{ij}\varepsilon_i(dW_{ij}/dt_i)(dT_{ij}/dt_i) \qquad (12b)$$

$$= - \Sigma_i\varepsilon_\iota(dU_i/dt_i)^2\ (dV_i/dU_i) - \Sigma_{ij}\varepsilon_i(dW_{ij}/dt_i)^2\ (dT_{ij}/dW_{ij}) \qquad (12c)$$

$$\leq 0 \qquad (12d)$$

The identical reasoning has already been given in Eq. (5a,b,c,d).

In the special case of two collective time scales, the present hairy model $E_3(V_i; \varepsilon T_{ij})$ is reduced to a couple set of first order equations Eqs. (4,11) similar to Grossberg-like short term retention and long term memory. Note that the proof shows that both tuning up synaptic gaps (spark plugs) and replacing wiring diagram for (car engine) can contribute to car engine efficiency and can be occurred at two time scales: $t_1= t$ and $t_2= \varepsilon t$ with $\varepsilon = 10^{-2}$ to be the diffusion ratio between the ion-mediated firing rates and the protein-mediated growths· Consequently, combining the Hopfield-like Eq. (4) in the fast collective time scale $t_1= t$ with the Rumelhart-like Eq. (6) in the slow collective time scale $t_2= \varepsilon t$ can produce for us the Grossberg-like equations coupled the short term memory adjustment with the long term memory learning mechanism. This concludes our unified convergence theory of neural network asynchronous computing.

IMAGE PROCESSING AND PATTERN CLASSIFICATION

In order to suppress the environmental clutter and to derive invariant information, the Human Visual System (HVS) passes the salient features, such as edges and orientations, before lowering the image resolution by low pass filters (feeding through layer-by-layer a Fukushima-like feed forward architecture using a local summations of the input image sequence). A visual system is sensitive to a moving object, and pays attention to changes with multiple looks (pointing and tracking towards the interesting or motion-detected part of scenery, Shown the eye trace in Fig. 5 Middle Right (Courtesy of A.L. Yarbus, in "Eye Movements and Vision," Plenum 1967). This kind of attentive image summation is believed to be important for better object template formation, and simultaneous feature extraction, as well as subsequent neural network pattern recognition. In Fig. 5, we have illustrated the importance in choosing a proper vector representation for images. To preserve the proximity relationship, French mathematician Peano has proved the letter N (in a 3x3 and an attentional 9x9 scanning) curves are everywhere uniform spiral scanning without being limited to the neighborhood of the origin. Semat et al. have also applied the Peano Z curves (formed by taking the significant bits of product values of pixel x and y coordinates) to represent a hierarchical QuarTree in order to search maps and study computer vision. A neuronic vector **V** has been recently proposed for neurocomputing of two dimensional images along the Peano scanning of the image space, because of the local relationship is preserved in carrying out Taylor series expansion with respect to the top-down design of the mini-max clustering energy of neural networks.

A video sequence of object seeing through the atmospheric turbulence has been taken downward through a wavy surface of turbulence simulated with a specific time-correlation scale. Then, a shifted-and-added technique was used in the second look to produce a sharp template of the submerged object. The second look was a regional re-summation that was re-done piecewise with respect to the identical set of imagery that has produced

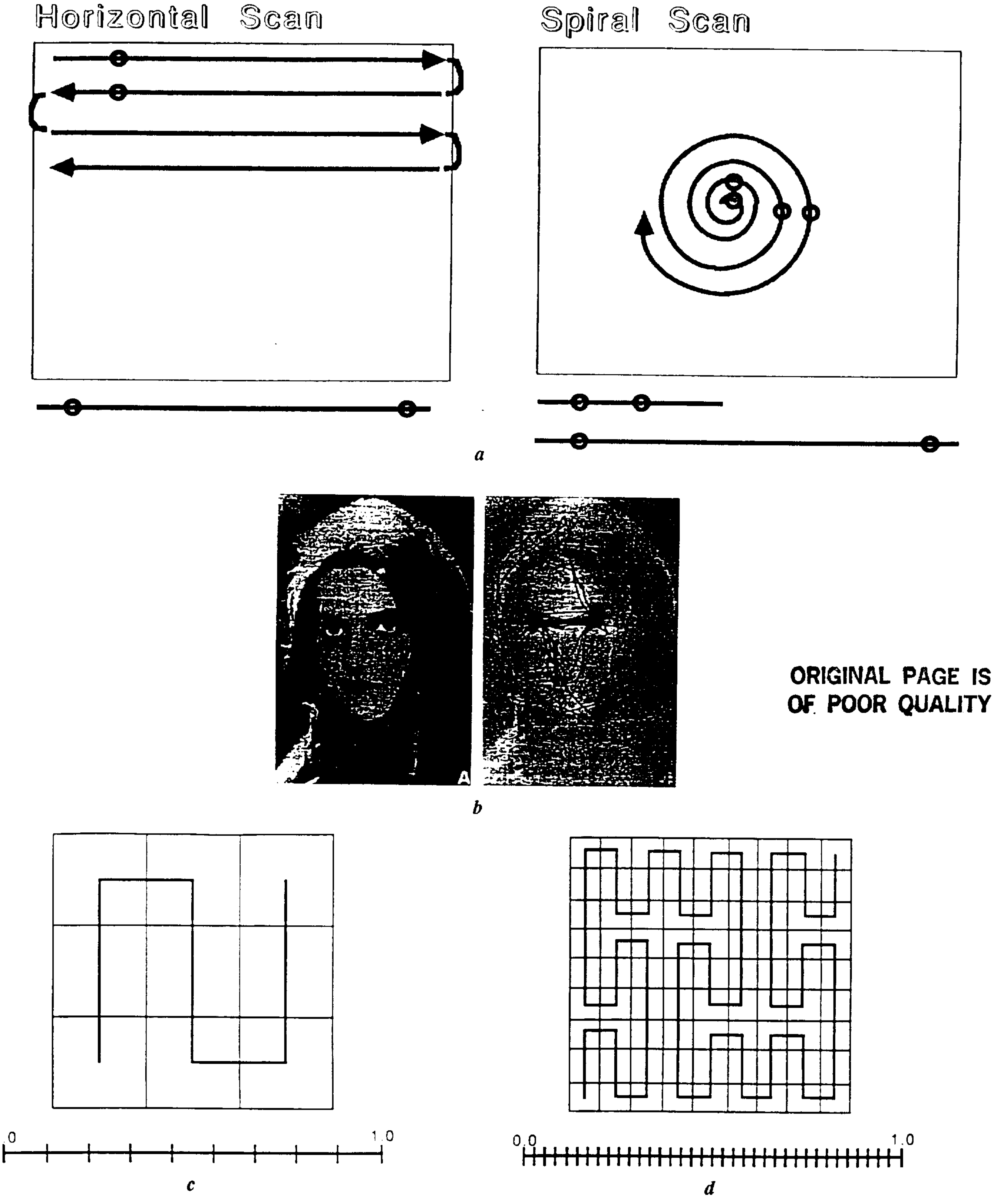


Figure 5. (a) Deficiency of horizontal scan and spiral scan in preserving the local proximity relationship. (b) Experimental measurement of human visual system scanning of interesting portions of a picture (courtesy of A. L. Yarbus). (c) Preferred Peano N-curve in 3x3 resolution grid. (d) Preferred Peano N-curve in 9x9 resolution grid.

in the first pass a blurred template which had a correct statistics of image pieces through the straightforward pointing-and-tracking summation of many frames (about 16 distorted fields) according to the centroid of the whole frame (Szu & Blodgett 1982) (c.f Distorted Fields, Object, Long Term Average, Centroid Correction). This effect had demonstrated the need of a smart sensor concept such as the eye which can see a weak star during an "instance of good seeing" (Szu et al. 1980) through the turbulent sky. On the contrary, the undiscriminating and dumb telescope camera can only produce a blurred picture of the weak star in the over exposed picture by the whole frame summation based on the straightforward pointing-and-tracking gimbal without any adaptive phase for turbulence medium phase correction mechanism.

Recently, a sequence of distorted imagery that consists of a training set of 15 samples of hand-written characters (each has 4 by 4 pixels, only trained to recognize 3 classes) has demonstrated the ability of generalization: recognize a new class of letter (Szu&Scheff 1989). This was done by means of critical feature extraction using the "mini-max concept" to discover by itself a new class of 5 more hand-written characters by analyzing the "intra-interclass clustering property" on the self-constructed feature space (c.f. Fig. 6 for 20 samples of 4 classes). This example used a table top computer, because the Gram-Schmidt orthogonal feature extraction was based on the associative memory employing the Fixed-Point Cycle Two Theorem (Szu, Scheff 1989). Such a procedure of parallel Gram-Schmidt constrained orthogonalization could be exceedingly usefully for a covert communication constrained by call signs and known scrambling instruction, because feature extraction by means of the straightforward projection is not permitted to obliterate critical portion of the signal. However, any practical construction of large set of orthogonal feature vectors could be subject to a realtime processing bottleneck. In this paper, the Fast Simulated Annealing (FSA) technique is adopted to alleviate the bottleneck problem.

Image processing by annealing techniques have been attempted (Geman & Geman,1984) (Smith et al. 1983) mainly for noise/distortion reduction. Neural networks have been recently applied to pattern recognition by Kohonen, Fukushima, Grossberg, Hopfield, etc.. White noise annealing and neural networks are combined through the Boltzmann Machine (Hinton, Sejnowski, Ackley, 1984) of which colored noise variant has been referred to as Cauchy Machine (Szu 1987) (Scheff &Szu 1987) (Takefuji & Szu 1989)

SPATIO-TEMPORAL IMAGERIES

A useful clutter rejection hypothesis is that man-made vehicles are designed to minimize the hydrodynamic drag via streamlined shapes and wheels while the natural environment of tree trunks is mainly vertical against the gravity (unpublished work of J. Landa, H.Szu). Thus, a sequence of imagery of land vehicles passing by bushes is considered, Fig. 7 (a). When a land vehicle moves by a tree, the partial occlusion of the vehicle by the tree trunk can be easily overcome by a properly pointing tracking, zooming, imaging on the moving vehicle. The image sequence can be averaged and threshold to get rid of the relative motion between the tree and the vehicle, Fig. 7 (b), together with the 9 by 9 scanning Peano curve. The centroid pointing and tracking of the vehicle is assumed to produce the averaged gray-scaled image $< I_c(x,y) >$

$$< I_c(x,y) > = \Sigma_j \; I_{j}(x+x_c,y+y_c)/ \text{ frames} \qquad (13)$$

where (x_c, y_c) is a vehicle local centroid coordinate. After a certain threshold, the obscuring effect of the tree and bush will be minimized. Fig.7 (describe the templates)

$$L_c(x,y) = \text{Threshold}(< I_c(x,y) >) \qquad (14)$$

Let the critical feature of the template class-c be denoted as $f_c(x,y)$. Then, the performance criterion is the minimum distance between the template of the c-class=1,2 together with the direction cosine in the numerator, and the maximum difference between feature vectors in the denominator. Thus, the mini-max filter energy is

$$E(f_c)= a \,\Sigma_{c \neq c'} (< f_c | f_{c'} >) + b \,\Sigma_{c=1,2,...} | f_c - \underline{I}_c |^2 + \Sigma_{c \neq c'} \; d \,/\, | f_c - f_{c'} |^2 \qquad (15)$$

where the coefficient of the direction cosine via the inner product < | > may be heavily weighted, e.g. by setting a = 10 (relative to b = 1, c=1, and d=10). The change of energy is defined as $\Delta E = E_{new} - E_{old}$.

CAUCHY MACHINE

The image space is 2-D; but the search space can be 1-D, provided that space-filling scanning technique is adopted here for mapping 2-D imagery space to 1-D search space and yet preserving the local neighborhood

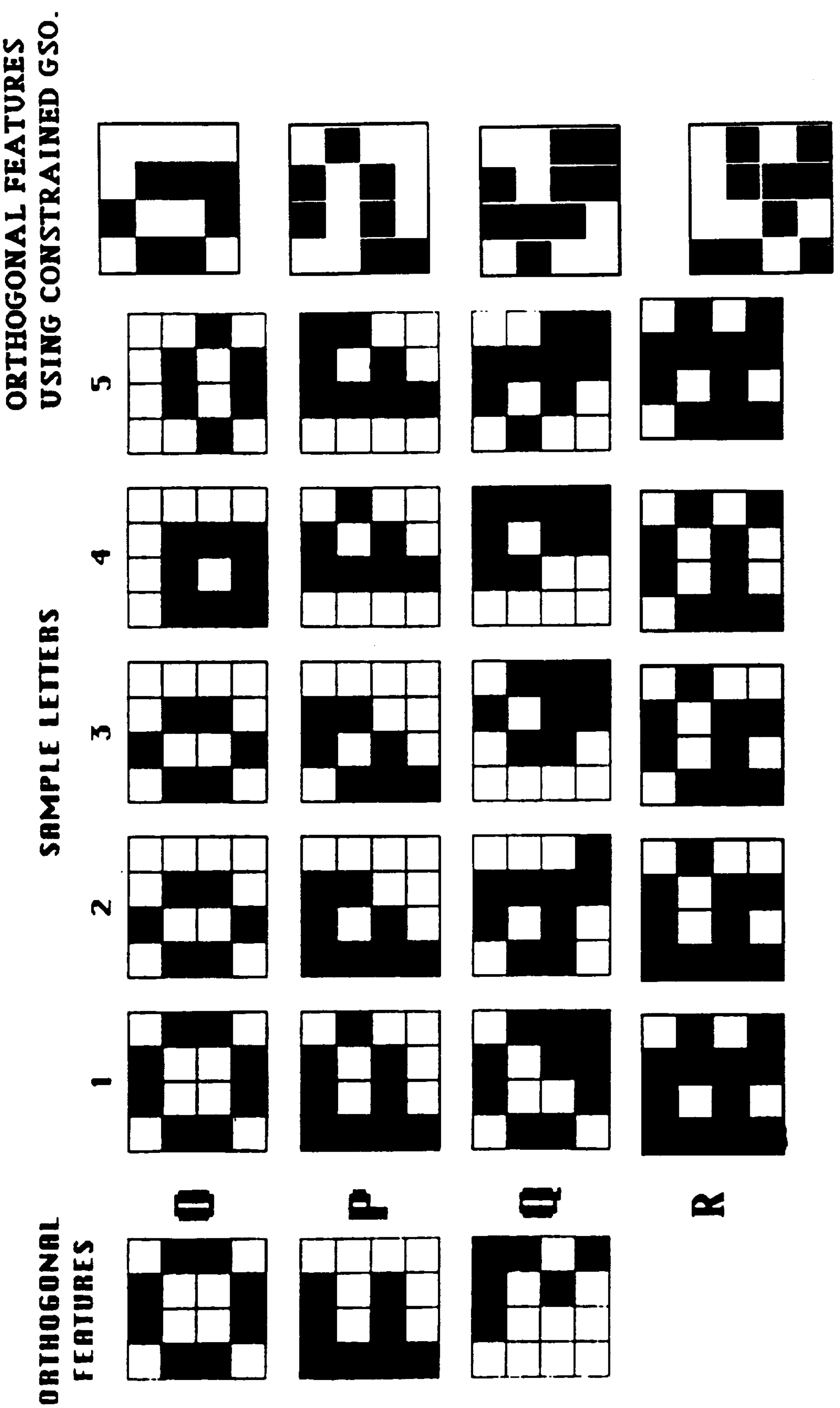
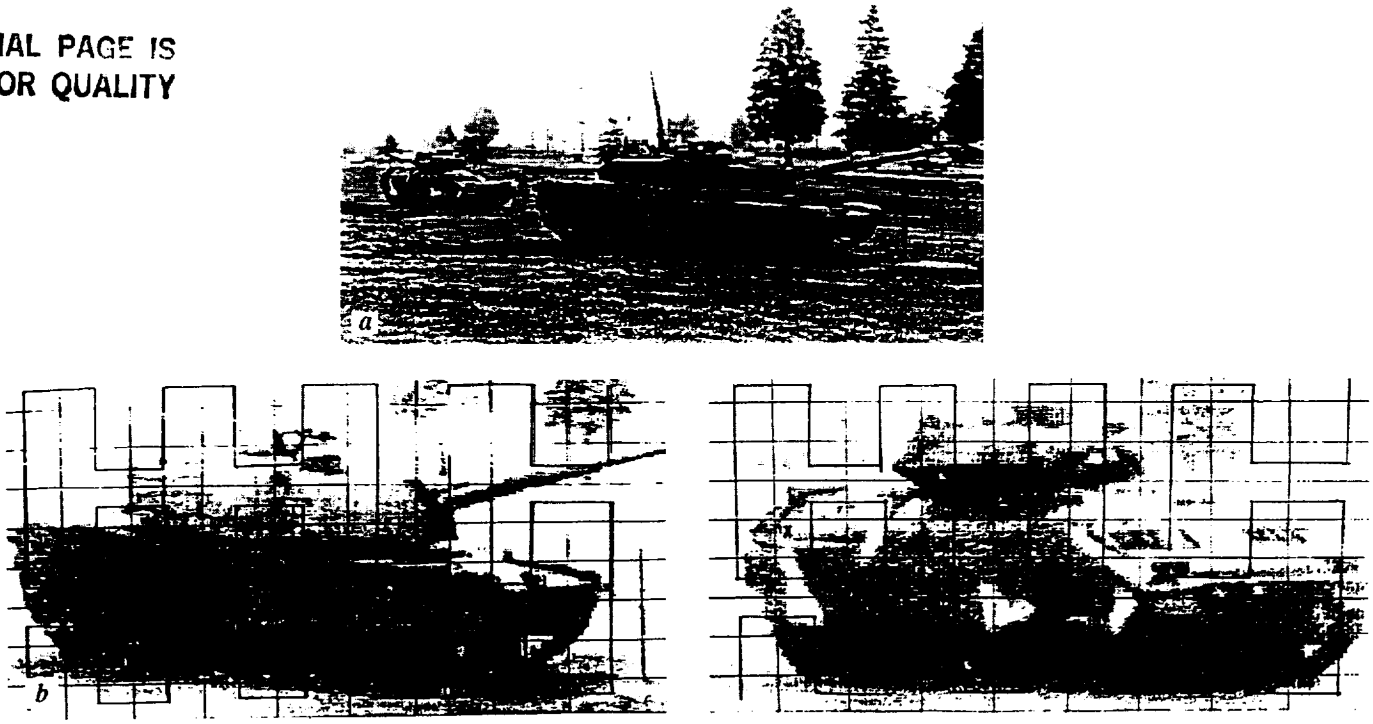
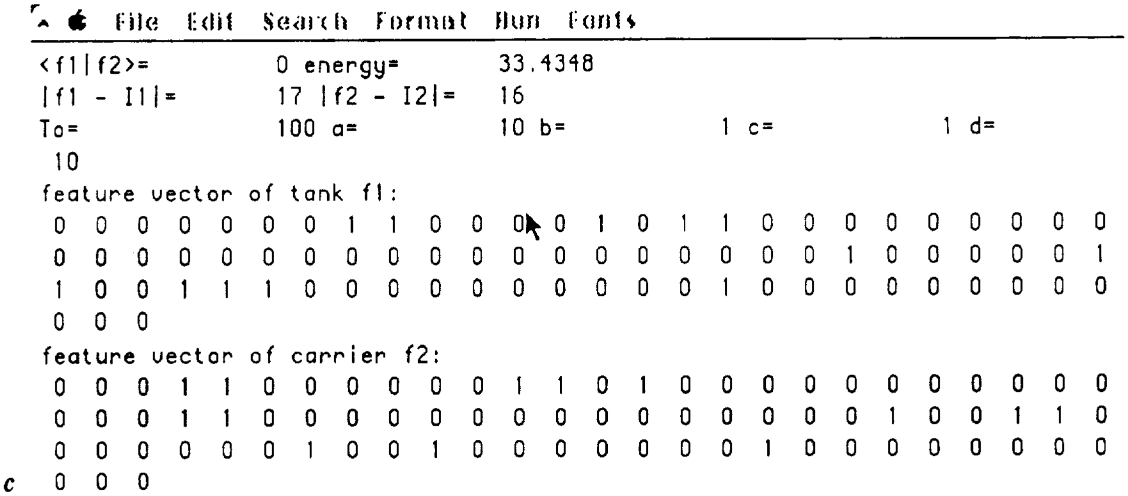
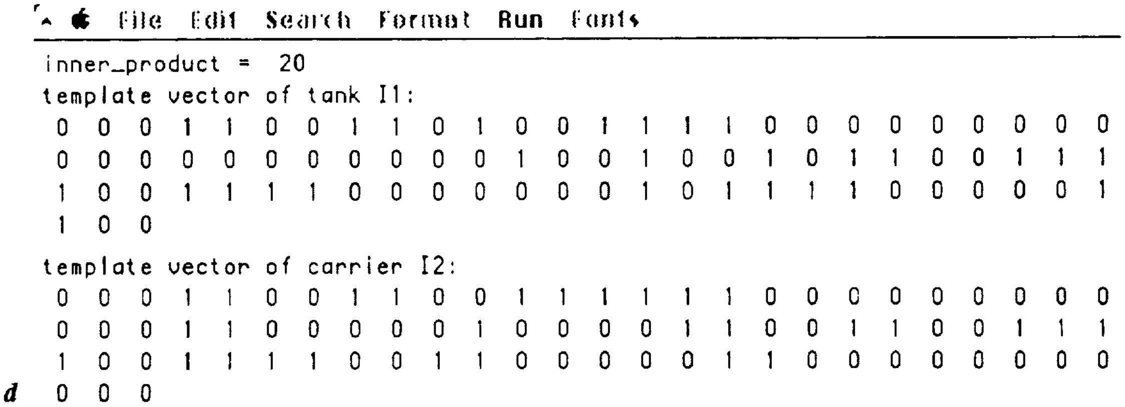


Figure 6. Hand-written character recognition by orthogonal feature extraction using constrained Gram-Schmidt orthogonalization (GSO) procedure.





```
<f1|f2>=          0 energy=      33.4348
|f1 - I1|=        17 |f2 - I2|=  16
To=               100 a=         10 b=           1 c=           1 d=
 10
feature vector of tank f1:
 0  0  0  0  0  0  0  1  1  0  0  0  0  1  0  1  1  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  1
 1  0  0  1  1  1  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0
 0  0  0
feature vector of carrier f2:
 0  0  0  1  1  0  0  0  0  0  0  1  1  0  1  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  1  1  0
 0  0  0  0  0  0  1  0  0  1  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0
 0  0  0
```



```
inner_product =  20
template vector of tank I1:
 0  0  0  1  1  0  0  1  1  0  1  0  0  1  1  1  1  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  1  0  0  1  0  0  1  0  1  1  0  0  1  1  1
 1  0  0  1  1  1  1  0  0  0  0  0  0  0  1  0  1  1  1  1  0  0  0  0  0  1
 1  0  0
template vector of carrier I2:
 0  0  0  1  1  0  0  1  1  0  0  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0
 0  0  0  1  1  0  0  0  0  0  1  0  0  0  0  1  1  0  0  1  1  0  0  1  1  1
 1  0  0  1  1  1  1  0  0  1  1  0  0  0  0  0  1  1  0  0  0  0  0  0  0  0
 0  0  0
```

Figure 7. (a) A snapshot of an imagery sequence. (b) 9x9 Peano scanning curve overlay upon two classes of image templates. (c) Binary template readout along the Peano scanning curves. (d) Simulation output for the automatic feature extraction.

relationship (Szu & Scheff 1990). In principle, the space-filling can be done to any desired degree of resolution, meaningful to original dynamic range and image pixel resolution.

The periodic 1-D search space is used for the 1-D infinite search space for the Cauchy probability governing a loaded dice. The random displacement X is equivalent to the following simple formula: using a random number n, normalized between [0,1], to generate a uniform angles: $(n - 0.5) \times \pi$, between $-\pi/2$ and $+\pi/2$.

$$X = T(t) \tan(\theta) \qquad (16)$$

(a) Generation Search States:

The new state x' is randomly chosen from the previous old state x by the Cauchy random number X, and then by the 1-1 mapping back to 2-D image domain:

Choose $f_1 = I_{c1}$.

Let f_2 be constructed from $\underline{I}_{c2}$.

$$G_T(x' | x'=x+X) = T(t) / \{[T(t)^2 + X^2] \pi\}; \qquad (17)$$

$$T(t) = T_0 / (1 + t)$$

where $T_0 = 100$ is arbitrary in this paper (in general may be estimated by the degree of freedom)

(b) Canonical Acceptance Criterion:

I_{c2} pixel toggling for $f_2(x')$ with $\Delta E < 0$ is accepted; the output state energy increase $\Delta E > 0$ is also accepted if the random number generated between [0,0.5] is less than the acceptance function]

$$P_T(\Delta E) = 1 / [1 + \exp(\Delta E / T(t))] \qquad (18)$$

Eq. (18) is similar to the Cauchy acceptance criterion (Takefuji & Szu 1989) when expressed in terms of the energy increment in a simulation by a serial process. To insure the mini-max property, Eq. (16), if $f_2(x')$ happens to be togged to be 0, we can reset $f_1(x')$ to $\underline{I}_1(x')$; otherwise, we change $f_2(x')$ back to $\underline{I}_2(x')$ and set $f_1(x') = 0$. The final data of f_1 and f_2 are given in Fig. 7 (d). The generating states, the accepting states, and the mini-max energy (Eq. (15)) are plotted in Fig. 8 which shows three segments of the ordinate (top segment: searching 9 x 9 states, middle segment: accepted 9 x 9 states, and bottom segment: the energy of the visited state) plotted with respect to the abscissa of 2000 time points in three minutes CPU time on a Macintosh II. Note that the scattering points about the accepted states is gradually narrowing down due to the Cauchy random walks but never completely because of the occasionally Cauchy random flights. Moreover, the energy occasionally goes up before it goes down, demonstrating the typical characteristics of simulated annealing.

CONCLUSION

In this paper, three convergence theorems for inhomogeneous and imperfect neuroprocessors have been given for neurocomputing for the first time, but we have not yet designed either VLSI chip or optoelectronic neurocomputer that can actually save the chip construction cost without the demand of precision timing among neuroprocessors. We have reviewed the state of art of neurocomputing, but we have not explicitly shown by simulations how to speed up the neural network training by the new perturbation expansion technique that can truncate the higher order Taylor series expansion of the mini-max neural networks. We have suggested that AI knowledge base expert system can be useful as a priori constraint upon which NI based smart neurosensor can look for, e.g. the tank track-barrel, and discover novel features for, e.g. personnel carriers, by means of the mini-max criterion for quasi-orthogonal features. We hope, but have not done, that the extracted features by NI can be used to expand AI expert system knowledge base. We propose to achieve these intelligence machine applications by working NI and AI together. In the future, we shall make no distinction between AI and NI by dropping the first letters A and N from both AI and NI and keeping only the common letter, the intelligence for the intelligent machine that we all wish to have in our shops.

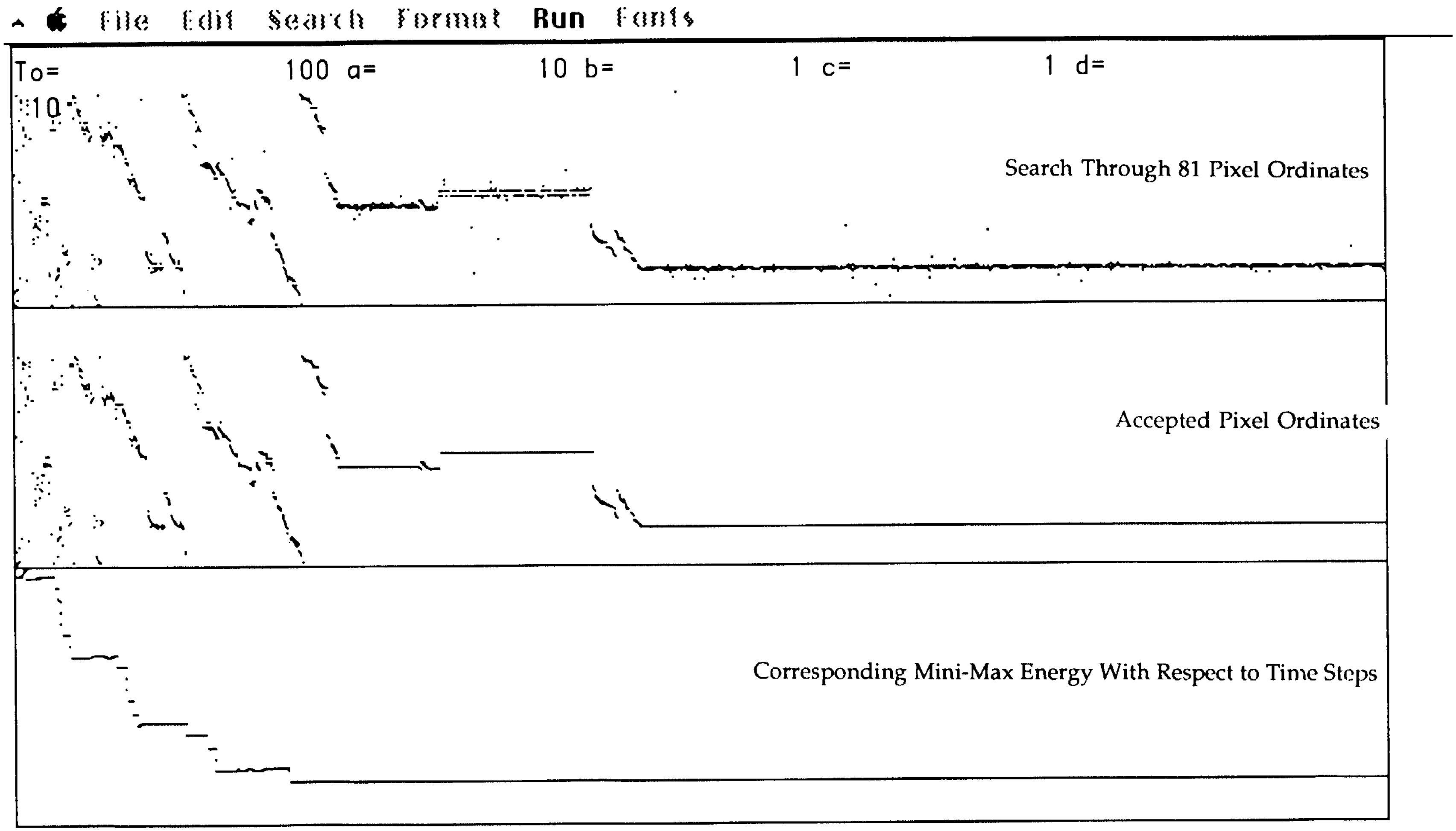
REFERENCES

Cohen, M.A. & Grossberg, S. (1983). Absolute Stability of Global Pattern Formation and Parallel Memory Storage by Competitive Neural Networks. IEEE Trans. Sys. Man Cybern. 13, p. 815-26

Duda, R.O. & Hart P.E. (1973). Pattern Classification and Scene Analysis. John Wiley & Sons, N. Y.

Geman, S., & Geman, D. (1984). Stochastic relaxation, Gibbs distribution and Bayesian restoration in images. *IEEE Trans Patt. Anal. Mach. Int. Vol. PAMI-6, pp.721-741,*

Hebb, D.O. (1949).The Organization of Behavior. John Wiley & Sons, New York



To= 100 a= 10 b= 1 c= 1 d=

10

File
Edit
Search
Format
Run
Fonts
Search Through 81 Pixel Ordinates
Accepted Pixel Ordinates
Corresponding Mini-Max Energy With Respect to Time Steps

Figure 8. Cauchy simulated annealing search for the mini-max global minimum energy. Three segments of the ordinate (top segment: searching 9x9 states, middle segment: accepted 9x9 states, and bottom segment: the energy of the visited state) are plotted with respect to the abscissa of 2000 time points in three minutes CPU time on a Macintosh II.

Hinton, G.E., Sejnowski, T.J., & Ackley, D.H. (1984). Boltzmann Machines: Constrained Satisfaction Networks that Learn. *CMU-CS-84-119, Carnegie Mellon Univ. May, 1984. "Parallel Distributed Processing, Vol. I , Vol.II Edited by J. McCelland, D. Rumelhart, PDP Group, MIT Press, 1986*

Hopfield, J.J. (1982). Neural Networks and Physical Systems with Emergent Collective Properties Like those of Two-State Neurons. Proc. Natl. Acad. Sci. USA Vol.79, pp. 2554-2558.

Kohonen, T (1984). Self-Organization and Associative Memory. Springer-Verlag, Berlin

McCulloch W.S. & Pitts, W. (1943). A logical Calculus of the Ideas Imminent in Nervous Activity. Bulletin of Mathematical Biophysics, 5, pp 115-133

Scheff, K., & Szu, H. (1987). 1-D Optical Cauchy Machine Infinite Film Spectrum Search. *Int. Conf.Neural Networks-87, P. III-673, San Diego*

Smith, W.E., Barrett, H.H., & Paxman, R.G. (1983). Reconstruction of objects from coded images by simulated annealing. *Optics Letters, Vol.8, pp 199-201*

Szu, H., Blodgett, J., & Sica, L. (1980). Local Instances of Good Seeing. *Optical Comm., Vol. 35, pp. 317- 322*

Szu, H., & Blodgett, J (1982). Self-reference Spatiotemporal Image-Restoration Technique.*J.Opt.Soc.Am., Vol.72, pp.1666-1669*

Szu, H., & Messner, R. (1986). Adaptive Invariant Novelty Filters. *Proc. IEEE, V.74, p.519*

Szu, H.,& Scheff, K. (1989). Gram-Schmidt Orthogonalization Neural Nets for Optical Character Recognition. *Int Joint Conference on Neural Networks, Vol. I, pp. 547-555, Washington D.C., June 18-22*

Szu, H. (1987). Fast Simulated Annealing. *In: "Neural Networks for Computing," AIP Conf. Vol. 15, pp. 420-425, Edited by J. Denker, Snow Bird U.T., 1987; Also, Phys. Letters A 122,p.157, Jun 8, 1987; Proc.IEEE, V. 75, p.1538.*

Tarkefuji, Y., & Szu,H. (1989). Parallel Distributed Cauchy Machine. *Int.Joint Conf. Neural Networks-89, p. I-529, Washington D.C. June 18-22*

Szu, H. (1989). Reconfigurable neural nets by Energy Convergence Learning Principle based on extended McCulloch and Pitts Neurons and Synapses. *Int.Joint Conf. Neural Networks-89, p. I-485, Washington D.C. June 18-22*

Szu, H. & Scheff, K (1990). Simulated Annealing Feature Extraction from Occluded and Cluttered Objects. *Int. Joint Conf.Neural Networks-90, p. II-76, Washington D.C. Jan.15-18,* Problem. *Int. Joint Conf.Neural Networks-90, p. I-317, Washington D.C. Jan.15-18*

Appendix A : Fast Simulated Annealing Algorithm (TRUE_BASIC Version)

```
DATA 4,5,8,9,11,14,15,16,17,38,41,44,46,47,50,51,52,53,56,5758,59,67,69,70,71,72,78,79        !input 81 Peano-scanning pixel#
DATA 4,5,8,9,12,13,14,15,16,17,30,31,37,42,43,46,47,50,51,52,53,56,57,58,59,62,63,69,70        !1= black feature Eq. (13)
DIM f1(81),f2(81),ave1(81),ave2(81),ft1(81),ft2(81)
MAT ave2 =0                                                            ! True_Basic  Matrix Operation
FOR n=1 to 29                                                          ! read an object into ave1, namely I1, Eq. (13)
  READ k
  LET ave1(k)=1
NEXT n
FOR m = 30 to 58                                                        ! read another object  into ave2, namely I2, Eq. (13)
  READ J
  LET ave2(J)=1
NEXT m
RANDOM                                                                 ! random number rnd generated [0,1]
FOR t=1 to tmax                                                        ! after initialize the display
  LET temp=To/(1+t)                                                    ! Fast Simulated Annealing cooling schedule
  LET theta=(rnd-.5)*Pi                                                ! uniform theta using the radian angle option
   LET dx=int(temp*tan(theta))                                         ! new pixel by T tan(theta), Eq. (15)
  LET xnew=mod(x+dx,82)                                                ! module for 81 scan pixels
  IF xnew=0 then LET xnew=81
  IF f2(xnew)=0 THEN
    LET ft2(xnew)=ave2(xnew)
    LET ft1(xnew)=0
  ELSE
    LET ft2(xnew)=0
    LET ft1(xnew)=ave1(xnew)
  END IF
  LET enew= 0
  LET denominator=0
  LET ef1=0
  LET ef2=0
  FOR n=1 to 81
     LET ef1=ef1+(ft1(n)-ave1(n))*(ft1(n)-ave1(n))
     LET ef2=ef2+(ft2(n)-ave2(n))*(ft2(n)-ave2(n))
      LET denominator=denominator+(ft1(n)-ft2(n))*(ft1(n)-ft2(n))
    LET enew = enew + ft1(n)*ft2(n)
  NEXT n
  LET enew= a*enew + b*ef1 + c*ef2 + (d/denominator)                  ! constants are typed into the code at run time
  IF enew<eold then
    MAT f2=ft2
    MAT f1=ft1
    LET eold=enew
    LET x=xnew
  END IF
  IF enew>=eold then
    IF (rnd*0.5)<(1/(1+exp((enew-eold)/temp))) then                    !hill climbing Eq. (16)
      MAT f2=ft2
      MAT f1=ft1
      LET eold=enew
     LET x=xnew
    END IF
  END IF
  PLOT POINTS :t,xnew+200
  PLOT POINTS :t,x+100
  PLOT POINTS :t,eold/2
NEXT t
```

Bounded-Time Fault-Tolerant Rule-Based Systems†

James C. Browne
Allen Emerson
Mohamed Gouda
Daniel Miranker
Aloysius Mok
Louis Rosier

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

Abstract

We introduce two systems concepts: bounded response-time and self-stabilization in the context of rule-based programs. These concepts are essential for the design of rule-based programs which must be highly fault-tolerant and perform in a real-time environment. The mechanical analysis of programs for these two properties will be discussed. We have also applied our techniques to analyze a NASA application.

Key words: rule-based programming, real-time, self-stabilization

1. Introduction

The operations and functions of systems that rely on the computer for real-time monitoring and control have become increasingly complex. However, there have been few attempts to formalize the question of whether rule-based systems can deliver adequate performance and be able to recover gracefully from transient faults in *bounded time*. In this paper, we provide a formal framework for answering these important questions.

The class of real-time programs that are investigated herein are called *equational rule-based (***EQL**) programs. An **EQL** program has a set of rules for updating variables which denote the state of the physical system under control. The firing of a rule computes a new value for one or more state variables to reflect changes in the external environment as detected by sensors. Sensor readings are sampled periodically. Every time sensor readings are taken, the state variables are recomputed iteratively by a number of rule firings until no further change in the variables can result from the firing of a rule.

EQL differs from the popular *expert system* languages such as **OPS5** in some important ways. Whereas the interpretation of a language like **OPS5** is defined by the *recognize-act cycle* (Forgy 1981), the basic interpretation cycle of **EQL** is defined by fixed point convergence, and no perference is given to

† Work supported partly by a research grant from the Office of Naval Research under ONR contract number N00014-89-J-1913, ONR contract number N00014-89-J-1472, and also partly by a grant from Texas Instruments Inc.

any enabled rule for firing when two or more are enabled. The differences with **OPS** reflect the goal of our research, which is not to invent yet another *expert system shell.* We want to investigate whether a rule-based program is sufficiently fast to react to a change in the environment, and whether it is sufficiently robust to recover from a corruption of its internal state.

2. Equational Rule-Based Programs: the EQL Language

A **EQL** program consists of a finite set of rules each of which has three parts:

(1) LHS: the left-hand-side of a multiple assignment statement,
(2) RHS: the right-hand-side of a multiple assignment statement, and
(3) EC: the enabling condition.

An enabling condition is a predicate on the variables in the program. (Whenever there is no ambiguity, we shall use the terms *enabling condition* and *test* interchangeably.) A rule is enabled if its test evaluates to true. A rule firing is the execution of the multiple assignment statement of an enabled rule. A multiple assignment statement assigns values to one or more variables in parallel. The format of a rule is:

<variable list> := <expression list> if <boolean expression>

The number of variables on the left hand side must be the same as the number of expressions on the right hand side, and the expressions must be side-effect free. The execution of a multiple assignment statement consists of the evaluation of all the RHS expressions, followed by updating the LHS variables with the values of the corresponding expressions.

An invocation of an equational rule-based program is a sequence of rule firings (execution of assignment statements whose tests are true). When two or more rules are enabled, the selection of which rule to fire is nondeterministic, i.e., up to the run-time scheduler.

The variables in a **EQL** program are either *input variables* (and their values are determined by sensor readings from the external environment at the beginning of each invocation of the program) or *internal variables*. Input variables do not appear on the left hand side of any assignment statement.

An equational rule-based program is said to have reached a *fixed point* with respect to an internal variable x when either:

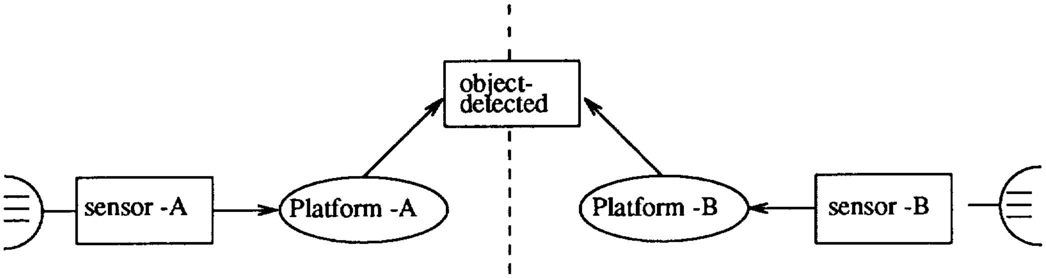
(1) none of the rules are enabled, or
(2) firing of any enabled rule will not change the value of x.

If a program reaches a fixed point with respect to all of its internal variables, then we say that the program has reached a fixed point. A **monitor-decide** cycle starts with the update of input (sensor) variables and this puts the program in a new state. A number of rule firings will modify the internal variables until the program reaches a fixed point. Depending on the starting state, a monitor-decide cycle may take an arbitrarily long time to converge to a fixed point if at all.

3. Bounded Response Time and Recovery from Abnormal States

To evaluate whether an **EQL** program is sufficiently fast to react to a change in the environment, we define the **response time** of an **EQL** program to be the maximum length of a monitor-decide cycle in any execution of the program. The response time of a program is infinite if it is possible for it to never reach a fixed point from a launch state in finite time.

Example



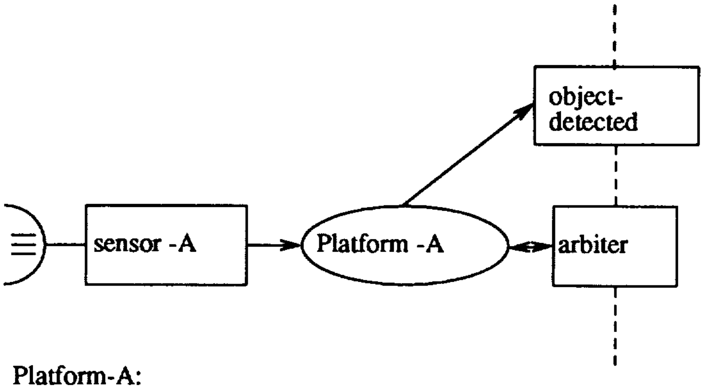
Initially, object-detected = false
System goals are:
(1) Set object-detected to true if either sensor detects object.
(2) In any computation, object-detected should reach a fixed point.
(3) The system should be self-stabilizing.

Fig 1: The system

Consider the parallel system shown in Figure 1 whose purpose is to determine whether an object has been detected by either of its two sensors. That is, the variable *object-detected*, initially set to *false*, is to be set to *true* by the code labelled *Platform_A* (*Platform_B*) whenever *sensor-A* (*sensor-B*) detects an object.

Figure 2 shows an attempt to implement this parallel system in terms of a rule-based program. Notice that this implementation does not reach a fixed point with respect to the variable *object-detected* since the value of *object-detected* will continually alternate between *true* and *false* if only one of the sensors detects an object, e.g., when sensor-A reports a 1 and sensor-B reports a 0. Thus this program has an infinite worst-case response time. Obviously, this is undesirable. One of our goals is to ensure that **EQL** programs for real-time applications must have bounded response time.

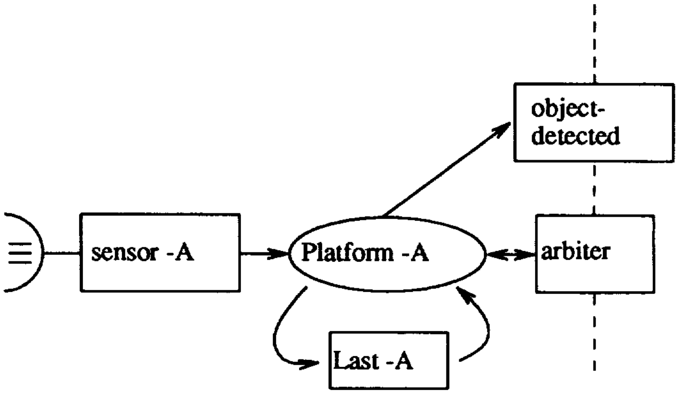
To evaluate whether an **EQL** program is sufficiently robust to recover from a transient upset, we are interested in the behavior of a program after some of its internal variables have been unintentionally modified by an external disturbance (e.g., a bit in dynamic memory may be flipped by cosmic ray). In such a case, we would like the program to be able to recover (through further execution) to a state that can be reached from some normal program execution path. A program that can always effect such a recovery is said to be self-stabilizing. It is also our goal to ensure that **EQL** programs for real-time applications are self-stabilizing.



```
Platform-A:
        object-detected, arbiter := true, B
                if arbiter = A ∧ sensor-A = 1
    []  object-detected, arbiter := false, B
                if arbiter = A ∧ sensor-A = 0

Platform-B:                (Summetric to A's code)
```

Fig 2: First attempt

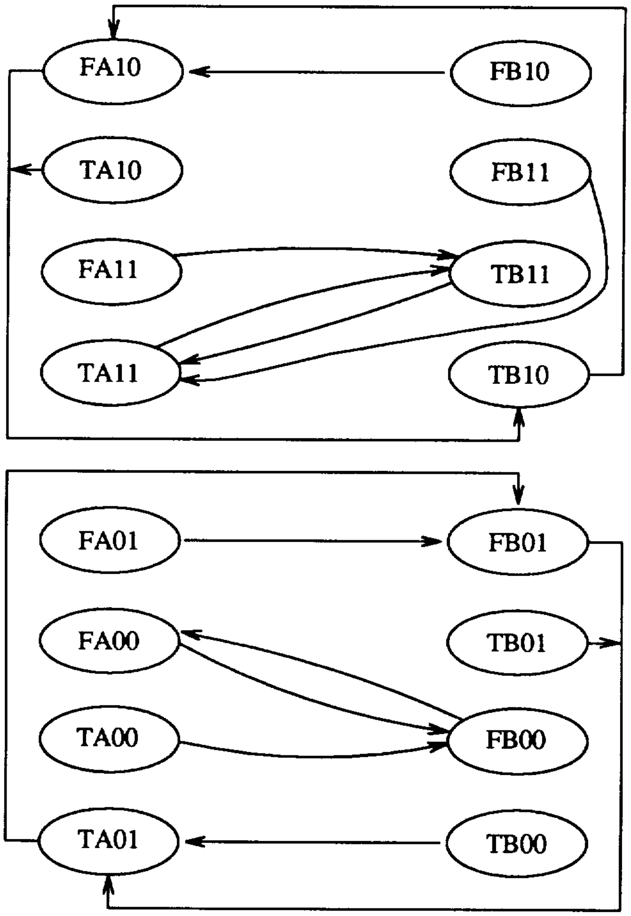


```
Platform -A:
        arbiter, object-detected, last-A := B, true, true
            if arbiter = A ∧ sensor-A = 1
    []  arbiter, object-detected, last-A := B, false, false
            if arbiter = A ∧ sensor-A =0 ∧ last-A = false
    []  arbiter := B
            if arbiter = A ∧ sensor-A = 0 ∧ last-A = true

Platform -B:            (symmetric)
```

Fig 3: Second attempt

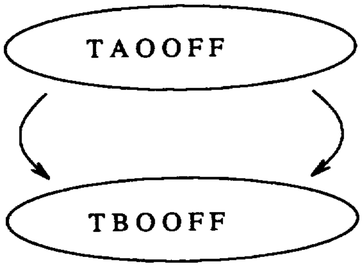
Figure 3 shows a second attempt to implement the system of Figure 1. Modulo a change in the sensor values (these are the system's input variables which are updated at the beginning of each sampling period), the implementation of Figure 3 will always reach a fixed point. Even though the implementation of Figure 2 does not always reach a fixed point, it is self-stabilizing, however, since with respect to any sensor reading any fixed point of *<arbiter, object-detected>* constitutes a normal state (see Figure 4).



state = <object-detected,arbiter,sensor-A,sensor-B>

Fig 4: State diagram

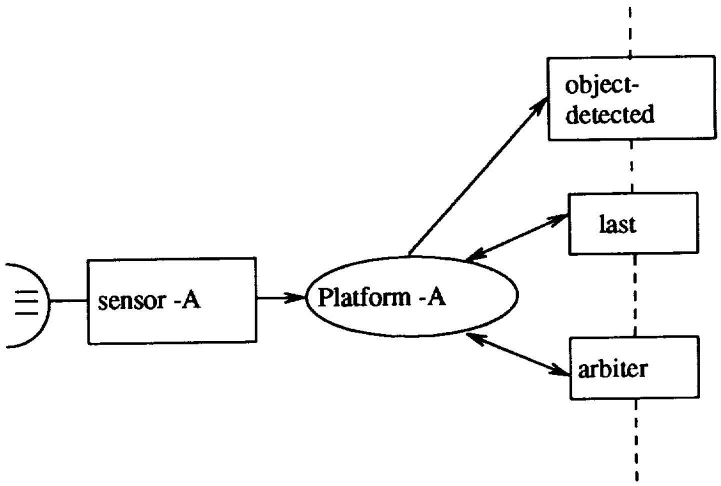
In contrast, the implementation of Figure 3 is not, however, self-stabilizing as the system in the abnormal state where *object-detected* = *true* , *last-A* = *last-B* = *false* , and *sensor-A* = *sensor-B* = 0 will be unable to recover (see Figure 5).



state = <object-detected,arbiter,sensor-a,sensor-B,last-A,last-B>

Fig. 5: Portion of the state diagram with
object-detected = true
last-A = last-B = false
sensor-A = sensor-B = 0

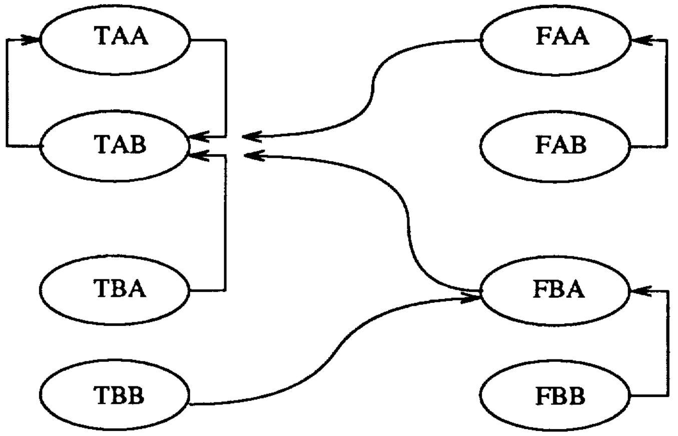
It is possible to design an implementation which always reaches a fixed point and is self-stabilizing. Such a program is shown in figure 6. A portion of its state-transition graph is shown in Figure 7.



```
Platform-A:
        arbiter, object-detected, last := B, true, A
            if arbiter = A ∧ sensor-A = 1
  []    arbiter, object-detected := B, false
            if arbiter = A ∧ sensor-A = 0 ∧ last = A
  []    arbiter := B
            if arbiter = A ∧ sensor-A = 0 ∧ last = B

Platform-B:          (symmetric)
```

Fig. 6: Last attempt



state = <object-detected,last,arbiter>

Fig 7: Portion of the state diagram
with sensor-A = 1 and sensor-B = 0

4. Formalization via State Space Representation

In order to formalize the response time and self-stabilization property of a rule-based program, we represent an **EQL** program in terms of its state space graph. The state space graph of an **EQL** program is a labeled directed graph $G = (V,E)$. V is a set of vertices each of which is labeled by a tuple: $(x_1, \ldots, x_n, s_1, \ldots, s_p)$ where each $x_{i,\ 1 \le i \le n}$ is a value in the domain of the i^{th} input sensor variable and each $s_{j,\ 1 \le j \le p}$ is a value in the domain of the j^{th} internal variable. We say that a rule is **enabled** at vertex u iff its test is satisfied by the tuple of variable values at vertex u. E is a set of edges each of which denotes the firing of a rule: an edge (u,v) connects vertex u to vertex v iff there is a rule R which is enabled at vertex u, and firing R will modify the internal variables to have the same values as the tuple at vertex v. Whenever there is no confusion, we shall use the terms state and vertex interchangeably.

A **path** in the state space graph is a sequence of vertices $v_1, \ldots, v_i, v_{i+1}, \cdots$, such that an edge connects v_i to v_{i+1} for each i. Paths can be finite or infinite. The length of a finite path $v_1, \ldots, v_k$ is $k-1$. A **simple path** is a path in which no vertex appears more than once. A **cycle** in the state space graph is a path $v_1, \ldots, v_k$ such that $v_1 = v_k$. A path corresponds to the sequence of states generated by a sequence of rule firings of the corresponding program.

A vertex v in a state space graph is said to be a **fixed point** if it does not have any out-edges or if all of its out-edges are self-loops, i.e., (v,v). Obviously, if the execution of a program has reached a fixed point, then every rule is either not enabled or its firing does not modify any of the variables.

An invocation of a rule-based program (a monitor-decide cycle) can be thought of as tracing a path in the state space graph. We say that a fixed point is an **end-point** of a state s if that fixed point is reachable from s. After a program reaches a fixed point, it will remain there until the sensor input variables are updated, and the program will then be invoked again in this new state. The states in which a program is invoked are called **launch states**. Formally, we define a launch state as follows:

(1) The initial state of a program is a launch state.

(2) A tuple obtained from an end-point (which is a tuple of input and internal variables) of a launch state by replacing the input variable components with any combination of input variable values is a launch state.

(3) A state is a launch state iff it can be derived from rule (1) and (2).

With respect to a state space graph, a tuple is a **normal state** if it appears on a path from a launch state to an end-point. Tuples which are not normal states are **abnormal states**. In the absence of faults or malfunctions, the variables of a program are by definition always in normal states. When the variables of a program are in an abnormal state (e.g., due to a hardware fault which arbitrarily writes over some of the internal variables), firing of an enabled rule may or may not bring the program back to a normal state.

To model the effect of faults, we define a **deviation function** B which maps each normal state s into a set $B(s)$ of tuples which may be normal or abnormal states. Intuitively, if a fault occurs when the program is in state s, then the program will be in one of the states specified by $B(s)$. We note that in the case where faults can have arbitrary effects on the variables, $B(s)$ may be the entire set of $n+p$-tuples. However, we expect that hardware techniques (e.g., error-correcting code) can be used to restrict the effect of faults so that $B(s)$ need not be very large for any normal state s. We say that a program is **deviation-bounded** with respect to the function B if for every normal state s, a fault will put the system into only a state in $B(s)$.

We are interested in systems which can recover automatically from transient faults. In particular, we define a **recovery function** R which maps each normal state s into $R(s)$, a subset of the set of normal states. The goal is to design systems such that if a transient fault occurs in state s and puts the system in an abnormal state, then further execution of the program will automatically bring the system back to a normal state in $R(s)$.

We say that a program is **self-stabilizing** with respect to the function R if for every normal state s, $R(s)$ contains all fixed points reachable from any state that the system may enter from s after a transient upset. This is the case if the end-points of all the states in $B(s)$ are in $R(s)$.

5. Bounded Response Time Analysis

With the state space representation, the response time of a program can be measured by the maximum length of a path from a launch state to a fixed point. (We assume that the evaluation of the enabling conditions and the right-hand-side of the assignment statements takes bounded time.) The problem of interest is to decide whether a fixed point can always be reached from a launch state on any sufficiently long path and if so, whether all these paths are shorter than a given bound. Conversion of path length to real time is possible by introducing a time metric on the paths of the state space graph. This depends on the specific architecture of an implementation and will not be discussed here.

If the state space of a program is infinite, it is in general impossible to decide if an **EQL** program has finite response time. For programs whose state spaces are finite, we can determine the response time by a brute-force state space search. However, this approach is impractical for larger programs since the number of states can grow exponentially fast. The determination of response time for finite state programs can be shown to be PSPACE-hard (Mok 1989). For certain classes of **EQL** programs, it is not necessary to check the complete state space in order to solve the decision problem. If the rules of a program fall under certain special forms (templates), the program is guaranteed to always reach a fixed point in a finite number of iterations, and there are efficient procedures to determine whether a set of rules falls under these special cases. Matching special forms alone, however, is not very effective since they may cover only a small portion of practical rule-based programs. We have invented a general analysis strategy which combines the power of special forms with program rewriting to combat the combinatorial explosion problem.

5.1 A Special Form

As an example, one of these special forms which is especially useful will be given below. First, some definitions are in order.

For ease of discussion, we define three sets of variables for an **EQL** program:

$L = \{ v \mid v$ is a variable appearing in LHS of some assignment statement $\}$
$R = \{ v \mid v$ is a variable appearing in RHS of some assignment statement $\}$
$T = \{ v \mid v$ is a variable appearing in EC of some assignment statement $\}$

Let $T = \{ v_1, v_2, \ldots, v_n \}$ and let $\bar{v}$ be the vector $<v_1, v_2, \ldots, v_n>$. With this definition, each test (enabling condition) in a program can be viewed as a function $f(\bar{v})$ from the space of $\bar{v}$ to the set { *true*, *false* }. Let f_a be the function corresponding to the test a and let V_a be the subset of the space of $\bar{v}$ for which the function f_a maps to *true*. We say that two tests a and b are **mutually exclusive** iff the subsets V_a and V_b of the corresponding functions f_a, f_b are disjoint. Obviously, if two tests are mutually exclusive, then only one of the corresponding rules can be enabled at a time.

Let L_x denote the set of variables appearing in LHS of rule x. Two rules a and b are said to be **compatible** if at least one of the following conditions holds:

(CR1) Test a and test b are mutually exclusive

(CR2) $L_a \cap L_b = \varnothing$

(CR3) Suppose $L_a \cap L_b \neq \varnothing$. Then for every variable v in $L_a \cap L_b$, the same expression must be assigned to v in both rule a and b.

We now give a special form of rules for which the decision problem can be solved efficiently.

- **Special Form A:**

A set of rules are said to be in special form A if all of the following three conditions hold.

(1) Constant terms are assigned to all the variables in L, i.e., $R = \varnothing$.
(2) All of the rules are compatible pairwise.
(3) $L \cap T = \varnothing$.

Theorem

An **EQL** program whose rules are in special form A will always reach a fixed point in a finite number of iterations.

The utility of special form A might seem quite limited since the rather restrictive conditions of the special form must be satisfied by the *complete* set of rules in a program. However, the main use of the special form in our analysis tools is not to identify special-case programs. We leverage the special form by applying it to appropriate subsets of rules in a program so as to find out if at least some of the variables must attain stable values in finite time.

5.2 The General Analysis Strategy

The exploitation of special forms in our general analysis strategy is best explained by an example.

Example

input: read(b, c)

```
1.    a1 := true  IF b = true ∧ c = true
2. [] a1 := true  IF b = true ∧ c = false
3. [] a2 := false IF c = true
4. [] a3 := true  IF a1 = true ∧ a2 = false
5. [] a4 := true  IF a1 = false ∧ a2 = false
6. [] a4 := false IF a1 = false ∧ a2 = true
```

For this program, $L \cap T \neq \varnothing$ and thus the rules are not of the special form described in the preceding section. However, observe that rules 1, 2 and 3 by themselves are of the special form A and that all the variables in these rules do not appear in the left-hand-side of the rest of the rules of the program and thus will not be modified by them. We can conclude that the variables $a1$ and $a2$ must attain stable values in finite time, and these two variables can be considered as *constants* for rules 4, 5 and 6 of the program. We can take advantage of this observation and *rewrite* the program into a simpler one, as shown below.

input: read($a1$, $a2$)

```
4. [] a3 := true  IF a1 = true ∧ a2 = false
5. [] a4 := true  IF a1 = false ∧ a2 = false
6. [] a4 := false IF a1 = false ∧ a2 = true
```

Note that $a\,1$ and $a\,2$ are now treated as input variables. This reduced program is of the special form since all assignments are to constants, L and T are disjoint, and all tests are mutually exclusive. Hence this program is always guaranteed to reach a fixed point in finite time. This guarantees that the original program must reach a fixed point in finite time.

There are in fact more special forms that can be exploited in the above fashion. Our general strategy for tackling the analysis problem is as follows.

Algorithm GIA

(1) Identify some subset of the rules which are of a special form (determined by looking up a catalog of special forms) and which can be treated independently. Rewrite the program to take advantage of the fact that some variables can be treated as constants because of the special form.

(2) If none of the special forms applies, identify an independent subset of the rules and check the state space for that subset to determine if a fixed point can always be reached. For this purpose, we use the model checking technique for the temporal logic **RTCTL** (Emerson, Mok, Srinivasan & Sistla 1989). Rewrite the program as in (1) to yield simpler ones if possible.

(3) Perform an analysis on each of the programs resulting from (1) or (2).

Intuitively, the general strategy described above allows us to use a special form in the induction step of a proof, by structural induction, that an **EQL** program has bounded response time. Thus relatively restrictive special forms may be exploited to analyze a much larger class of programs.

6. Self-stabilization via Program Transformation

In general, it is not always possible to implement an application by a self-stabilizing program (Gouda, Howell & Rosier 1988). However, for a special class of **EQL** programs, it is always possible to transform a program in this class into an equivalent one which is deviation-bounded with respect to a function B and self-stabilizing with respect to a function R where: for any normal state s, $B(s)$ contains all tuples whose input-variable components agree with s, and $R(s)$ contains all end-points of s. Notice that this $B(s)$ requires the input variables remain unchanged by a transient upset. This may not be necessary if the input variables can be restored by repeating the sensor readings after the transient has subsided.

6.1 Acyclic Programs

We now consider the class of rule-based programs where each program P satisfies the following four conditions. Programs in this class are called acyclic programs.

[1] Syntax: Program P is defined by a finite set of assignment statements each of which is of the form:

$$x_i := B_i(x) \text{ if } C_i(x);$$

where x_i is a variable in P, x is the vector of all variables in P (thus x_i is a component of x), $B_i(x)$ is an expression of the same type as x_i and $C_i(x)$ is a boolean expression over the variables in P. Each internal variable x_i has an initial value denoted $\hat{x}_i$.

[2] Semantics: Consistent with **EQL** semantics, the assignment statements in P are executed one at a time. The order in which the statements are executed is arbitrary provided that each statement is executed infinitely often.

[3] Well-Formedness: For every pair of statements with the same left side

$$x_i := B_i(x) \text{ if } C_i(x);$$

$$x_i := D_i(x) \text{ if } E_i(x);$$

and for every value s of vector x, we have

$$C_i(s) \wedge E_i(s) \rightarrow B_i(s) = D_i(s)$$

[4] Acyclicity: The dependency graph of a program P is a directed graph where each node represents an internal variable of P, and there is an edge from node x_i to node x_j iff x_i appears in the right side of an assignment statement (in P) whose left side is x_j. A program is called **acyclic** iff its dependency graph is acyclic. P must be acyclic.

Acyclic programs can be shown to obey the following two theorems.

Theorem

Executing the statements of an acyclic program starting from any normal state leads eventually to a fixed point.

Theorem

If a program is acyclic, then it will be recognized by the GIA algorithm with special form A as having bounded response time.

6.2 Self-Stabilization

For an acyclic program P, it can be shown that for each pair s_1 and s_2 of distinct fixed points of P, there is at least one input variable whose value in s_1 is different from its value in s_2. In other words, the fixed point that an acyclic program P can reach starting from any state s, depends solely on the values of the input variables in s and not on the values of internal variables in s. Therefore, if program P is at a fixed point and the values of one or more internal variables change due to some failure, P is guaranteed to converge back to the same fixed point (as long as the values of its input variables remain unchanged).

6.3 Implementation

Given an acyclic program, we shall transform it into another program which is self-stabilizing. The transformed program must also implement the semantics of the original program.

A program P is said to **implement** program Q iff the following conditions hold:

[1] Programs P and Q have the same input variables.

[2] Each internal variable of Q is an internal variable of P.

[3] Each fixed point of P is a reachable fixed point of Q, and, if Q has a reachable fixed-point then P has a fixed point.

Theorem

For each acyclic program Q, there is an acyclic self-stabilizing program P that implements Q.

Proof: (by construction)

Every statement in Q is a statement in P. For each internal variable x_i in Q, do the following. Let all the statements in Q with x_i on the left side be:

$$x_i = B_i(x) \text{ if } C_i(x)$$

$$\ldots$$

$$x_i = D_i(x) \text{ if } E_i(x)$$

Then add the following statement to P:

$$x_i := \hat{x}_i \text{ if } \overline{C_i(x)} \wedge \cdots \wedge \overline{E_i(x)}$$

Here $\hat{x}_i$ is the initial value of the variable x_i. The resulting program P is acyclic, self-stabilizing and implements Q.

In fact, for acyclic programs, we can show that the end-point of any launch state is unique and depends only on the value of the input variables. Let $F(x_1, \ldots, x_n)$ be the function which maps a launch state into its end-point.

Theorem

For each acyclic program P, the self-stabilizing version of P has a recovery function given by: $R(x_1, \ldots, x_n, s_1, \ldots, s_p) = F(x_1, \ldots, x_n)$.

7. Application to NASA Program

We have taken a NASA application: the Cryogenic Hydrogen Pressure Malfunction Procedure of the Space Shuttle Vehicle (SSV) Pressure Control System (Helly 1984) and translated it directly into an EQL program. This program has 36 rules and 31 internal variables. We have mechanically verified that this program has bounded response time by using the GIA algorithm and Special Form A. The analysis took under 1 second of time on a SUN 3® workstation, whereas a brute-force state space search took over a week, even for a 20-rule subset of the program. We also determined that this program is not self-stabilizing but is acyclic. The transformation described in the paper was used to convert it into a self-stabilizing program.

8. Conclusion

In this paper, we have introduced two systems concepts: the notion of response time for a rule-based program and the application of self-stabilization to rule-based programs. These two concepts are essential for the design of rule-based programs which must be highly fault-tolerant and perform in a real-time environment. The mechanical analysis of programs for response-time boundedness was discussed. We also gave an algorithm to convert non-self-stabilizing programs to self-stabilizing ones for the class of acyclic programs. These concepts have been applied to a NASA program, the Cryogenic Hydrogen Pressure Malfunction Procedure of the Space Shuttle Vehicle (SSV) Pressure Control System.

Much work remains to be done, such as implementation techniques for realizing a given deviation function, more powerful techniques for determining response time and transformation techniques for ensuring self-stabilization.

Bibliography

Emerson, E. A., Mok, A, Srinivasan, J. & Sistla, A. P. (1989). Quantitative Temporal Reasoning. *Proceedings of Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, France, June 1989.

Forgy, C. L. (1981). *OPS5 User's Manual.* Department of Computer Science, Carnegie-Mellon University, Tech. Rep. CMU-CS-81-135, July 1981.

Gouda, M., Howell, R. & Rosier, L. (1988). The Instability of Self-Stabilization. *manuscripted, submitted for publication, 1988*

Helly, J. J. (1984). Distributed Expert System for Space Shuttle Flight Control. *Ph.D. Dissertation*, Department of Computer Science, UCLA, 1984.

Mok, A. K. (1989). Formal Analysis of Real-Time Equational Rule-Based Programs. *Proceedings, 10th Real Time Systems Symposium*, Los Angeles, December 5-7, 1989, pp. 308-318.

Appendix

The following is the **EQL** version of a NASA application: the Cryogenic Hydrogen Pressure Malfunction Procedure of the Space Shuttle Vehicle (SSV) Pressure Control System. This program was shown to have bounded response time by using the GIA algorithm with Special Form A. It was also transformed into a self-stabilizing version.

```
* SSV Cryogenic Hydrogen Pressure Malfunction Procedure
* Non-self-stabilizing version
*
* Translated into EQL by Albert Mo Kim Cheng
* 36 rules
```

```
PROGRAM cryov63a;

RULES
  v63a2 := true IF (v63a1a = true)
[]v63a4 := true IF (v63a1c = true) AND (v63a3 = true)
[]v63a6 := true IF (v63a1c = true) AND (v63a3 = false) AND (v63a5 = true)
[]v63a7 := true IF (v63a6 = true)
[]v63a9 := true IF (v63a1c = true) AND (v63a3 = false) AND (v63a5 = false) AND
                  (v63a8 = true)
[]v63a10 := true IF (v63a9 = true)
[]v63a14 := true IF ((v63a12 = true) OR ((v63a12 = false) AND (v63a13 = true)))
[]v63a15 := true IF (v63a1c = true) AND (v63a3 = false) AND (v63a5 = false) AND
                   (v63a8 = false) AND (v63a11 = false) AND
                   (v63a12 = true) AND (v63a14 = true)
[]v63a18 := true IF (v63a1c = true) AND (v63a3 = false) AND (v63a5 = false) AND
                   (v63a8 = false) AND (v63a11 = true) AND
                   (v63a16 = true) AND (v63a17 = true)
[]v63a19 := true IF (v63a1c = true) AND (v63a3 = false) AND (v63a5 = false) AND
                   (v63a8 = false) AND (v63a11 = true) AND (v63a16 = true)
[]v63a20 := true IF (v63a1c = true) AND (v63a3 = false) AND (v63a5 = false) AND
                   (v63a8 = false) AND (v63a11 = true) AND
                   (v63a16 = true) AND (v63a17 = false)
[]v63a21 := true IF ((v63a19 = true) OR (v63a20 = true))
[]v63a24 := true IF (v63a22 = true) AND (v63a14 = false) AND (v63a12 = true) AND
                   (v63a11 = false) AND (v63a8 = false) AND (v63a5 = false) AND
                   (v63a3 = false) AND (v63a1c = true)
[]v63a25 := true IF (v63a22 = false) AND (v63a14 = false) AND (v63a12 = true) AND
                   (v63a11 = false) AND (v63a8 = false) AND (v63a5 = false) AND
                   (v63a3 = false) AND (v63a1c = true)
[]v63a27 := true IF (v63a26 = true) AND (((v63a23 = true) AND (v63a1b = true)) OR
                                        (v63b7 = true))
[]v63a28 := true IF ((v63a25 = true) OR (v63a15 = true))
[]v63a30 := true IF (((v63a1b = true) AND (v63a23 = true)) OR (v63b7 = true)) AND
                   (v63a26 = false) AND (v63a29 = true)
[]v63a33 := true IF (v63a32 = false) AND (v63a31 = true) AND (v63a29 = false) AND
                   (v63a26 = false) AND (v63a23 = true) AND (v63a1b = true)
[]v63a35 := true IF (v63a32 = true) AND (v63a31 = true) AND (v63a29 = true) AND
                   (v63a26 = false) AND (v63a23 = true) AND (v63a1b = true)
[]v63a36 := true IF (v63a34b = true) AND (v63a31 = false) AND (v63a29 = false) AND
                   (v63a26 = false) AND (v63a23 = true) AND (v63a1b = true)
[]v63a37 := true IF (v63a30 = true) AND (v63a33 = false) AND (v63a35 = false) AND
                   (v63a38 = true)
[]v63a38 := true IF (v63a34a = true) AND (v63a31 = false) AND (v63a29 = false) AND
                   (v63a26 = false) AND (v63a23 = true) AND (v63a1b = true)
[]v63a39 := true IF (v63a36 = true)
[]v63a42 := true IF (v63a41 = true) AND (v63a40 = true) AND (v63a23 = false) AND
                   (v63a1b = true)
[]v63a43 := true IF (v63a41 = false) AND (v63a40 = true) AND (v63a23 = false) AND
                   (v63a1b = true)
[]v63a44 := true IF (v63a42 = true) OR (v63a47 = true)
[]v63a47 := true IF (v63a46 = true) AND (v63a45 = true) AND (v63a40 = false) AND
                   (v63a23 = false) AND (v63a1b = true)
[]v63a47 := true IF (v63a46 = true) AND (v63a45 = true) AND (v63b8 = true)
[]v63a48 := true IF (v63a46 = false) AND (v63a45 = true) AND (v63a40 = false) AND
                   (v63a23 = false) AND (v63a1b = true)
[]v63a48 := true IF (v63a46 = false) AND (v63a45 = true) AND (v63b8 = true)
[]v63a50 := true IF (v63a49b = true) AND (v63a45 = false) AND (v63a40 = false) AND
                   (v63a23 = false) AND (v63a1b = true)
[]v63a50 := true IF (v63a49b = true) AND (v63a45 = false) AND (v63b8 = true)
[]v63a51 := true IF (v63a50 = true)
[]v63a52 := true IF (v63a49a = true) AND (v63a45 = false) AND (v63a40 = false) AND
                   (v63a23 = false) AND (v63a1b = true)
[]v63a52 := true IF (v63a49a = true) AND (v63a45 = false) AND (v63b8 = true)
[]v63a53 := true IF (v63a52 = true)

END.
```

A KNOWLEDGE-BASED SYSTEM WITH LEARNING FOR COMPUTER COMMUNICATION NETWORK DESIGN

by

Samuel Pierre
Télé-université, Université du Québec
4835 Christophe-Colomb, Montréal, Qué. H2J 4C2
Tel: (514) 522-3540 Fax: (514) 522-3540

Hai Hoc Hoang
École Polytechnique de Montréal
Campus de l'Université de Montréal
C.P. 6079 - Succ. "A", Montréal, Qué. H3C 3A7

Evelyne Hausen-Tropper
Département de Mathématiques et Informatique, UQAM
C.P. 8888 - Succ. "A", Montréal, Qué. H3C 3P8

ABSTRACT

Computer communication network design is well-known as complex and hard. For that reason, the most effective methods used to solve it are heuristic. In this paper, we list weaknesses of these techniques, and present a new approach based on artificial intelligence for solving this problem. This approach is particularly recommended for large packet-switched communication networks, in the sense that it permits to ensure high degree of reliability, and offers a very flexible environment dealing with many relevant design parameters as link cost, link capacity and message delay.

KEYWORDS: knowledge-based system, communication network design, inductive learning.

1. INTRODUCTION

A computer communication network is generally modelled as a valued graph whose nodes represent computers and arcs communication links [2, 3]. Before implementing protocols allowing the operation of a network, we must determine the manner whose nodes are linked between them and the capacity of each link. Such a problem is known in the literature as the topological design of computer communication networks [2, 8, 13].

This paper proposes a knowledge-based system with inductive learning for solving this problem. It is organized as follows: section 2 sets up background for the topological design problem and underlines some weaknesses of conventional methods; section 3 puts forward the architecture and the running of the knowledge-based system; section 4 deals with the knowledge organization within the system; section 5 conceptualizes the inductive learning module and states the learning algorithm; section 6 summarizes some results and makes concluding remarks.

2. THE TOPOLOGICAL DESIGN PROBLEM

In this section, we first present prerequisite definitions and notations, a formulation of the topological design problem, and finally the conventional methods used to solve it.

2.1 DEFINITIONS AND NOTATIONS

Let us consider a set of nodes N and a set of edges A connecting these nodes. Let n be the cardinality of N and m the cardinality of A. A "topology" is an undirected graph G=(N,A), where each edge represents a full duplex link with a given capacity, expressed in bits per second (bps).

There are [n(n-1)/2] possible links between all pairs of nodes. This number is denoted by m_{max}.

So, the basic characteristics of a topology are its topological configuration materialized by A, which can be represented by a binary characteristic vector $t=(t_k)$, $k=1,2,...,m_{max}$, and its capacity assignment. For convenience, we shall use i to denote the i-th node and k=(i,j) the edge joining node i and node j, with $i,j = 1,2,...,n$, $i \neq j$, and $k = 1,2,...,m_{max}$. Such a numbering scheme can easily be devised. Note that:

$$\sum_k t_k = m \; .$$

It follows that various topological configurations can be obtained by varying the set of links.

For a given topology, each link k of the topological configuration $t=(t_k)$ is assigned a capacity C_k, such that $t_k=0$ implies $C_k=0$. $C=(C_k)$ denotes the capacity vector associated with the topology. Consequently, a topology will denoted by (t,C). Each link k of t is associated with a cost D_k which is a function of its capacity C_k:

$$D_k = d_k(C_k) \qquad (1)$$

In reference to the running network, all information or message to be transmitted is first broken in small parts called "packets". Independently passing from one node to another, these packets are reassembled at the destination: this is the packet-switching principle [11].

Let $1/\mu$ be the average packet length expressed in bits/packet and γ_{ij} the required traffic in packets/second from source i to destination j. The traffic requirement r_{ij}, expressed in bits/second, can be defined as follows:

$$r_{ij} = \gamma_{ij}/\mu \qquad (2)$$

Then the traffic matrix is $R=(r_{ij})$, i, j = 1,2,...,n, with $i \neq j$.

In order to satisfy the traffic requirements, it is first necessary to choose a routing strategy. The choice is generally motivated by computational considerations and should make the link flow computation relatively easy. If we denote by $f_k^{(p,q)}$ the flow in bps on link k produced by packets travelling from source p to destination q, the total flow f_k in link k is given by:

$$f_k = \sum_{\substack{p=1 \\ (p \neq q)}}^{n} \sum_{q=1}^{n} f_k(p,q) \qquad (3)$$

Consequently, the overall network flow can be represented by a flow vector:

$$f = (f_k) \qquad (4)$$

For a given topological configuration, f is uniquely determined by the routing strategy. Note that $C_k=0$ implies $f_k=0$. Thus, $t_k=0$ implies $f_k=0$.

The routing problem concerns the choice of the best path, according to a given criterion, for traffics from a source to a destination, provided that there exist multiple routes between all pairs of nodes. Such a situation materializes the concept of K-connectivity often used as a network reliability metric.

There are two types of connectivity: the edge-connectivity C_e, and the node-connectivity C_n. The edge-connectivity between two nodes i and j can be defined as the minimum number of edges whose removal will disconnect these two nodes. If we call edge-disjoint paths the paths which have no edges in common, then such an edge-connectivity is equivalent to the number of edge-disjoint paths between the two considered nodes. So, the edge-connectivity of a network is the minimum of the edge-connectivities amongst all pairs of nodes, that is, the number of edge-disjoint paths connecting the most critically connected pair of nodes.

Similarly, the node-connectivity between two nodes i and j is the minimum number of nodes which must be removed from the network to disconnect these two nodes. If we take the minimum node-connectivity over all pairs of nodes, we obtain the node-connectivity of the network, C_n.

If we denote by d the degree of a network, that is, the minimum degree of all nodes, it can be shown that $C_n \leq C_e \leq d$. So, for design purposes and for a given degree of connectivity, the node-connectivity C_n is more demending than the edge-connectivity.

Packets take time for travelling from source i to destination j. The average packet delay from i to j, is denoted by z_{ij}. The overall average delay T can be generally expressed as follows:

$$T = \frac{1}{\gamma} \sum_{\substack{i=1 \\ (i \neq j)}}^{n} \sum_{j=1}^{n} \gamma_{ij} \, z_{ij} \qquad (5)$$

where γ is the total traffic in the network which can be obtained by summing the γ_{ij}'s.

Based on a set of simplifying assumptions, a useful and easily computable expression for the overall average delay has been derived [2]:

$$T = \frac{1}{\gamma} \sum_{k \,\varepsilon\, A} \frac{f_k}{C_k - f_k} \qquad (6)$$

So, the overall average delay T appears as a function of link capacities C_k and link flows f_k, for all links included in the topological configuration which is considered.

2.2 PROBLEM FORMULATION

The network topological design problem can be formulated as follows [2, 4, 7, 8]:

Given:

. Switching node locations
. Traffic requirements $R=(r_{ij})$
. Capacity options and associated costs for all potential links
. Maximum overall average delay allowed T_{max}

$$\text{Min } D = \sum_{k} d_k(C_k) \qquad (7)$$

Over:

. Topological configuration t
. Capacity vector C
. Flow vector f

Subject to:

. $f \leq C$ (component wise)
. t is a K-connected topological configuration, $2 \leq K \leq n-1$
. $T = \frac{1}{\gamma} \sum_{k \varepsilon A} \frac{f_k}{C_k - f_k} \leq T_{max}$ (8)

This problem is known to be NP-hard [5, 6]. The first difficulty arises from the combinatorial character of link selection which involves some explosion risk.

Another major difficulty is the nonlinearity of relevant functions such as communication link costs D, and the average packet delay T. For that reason, only local optima are guaranteed by Kuhn-Tucker conditions [2].

Finally, link capacities are only available in some discrete values as 2400, 4800,9600, 19200, 50000 bps, etc. That constitutes a nontrivial problem which cannot be efficiently solved by discrete programming techniques, because of the size of the problem [2].

2.3 CONVENTIONAL METHODS

Taking into account the previous considerations, it is not suitable to search for an exact solution. Only approximate methods are recommended for finding realistic and suboptimal solutions. In fact, the combinatorial nature of this problem suggests the use of heuristics for attempting to reduce the search space of candidate topologies.

Most of conventional procedures use heuristics, and produce suboptimal solutions. They essentially correspond to search procedures which optimize network structure by sequentially changing small parts of a larger network [2, 9, 10, 12, 14].

In the case of small size networks (about 30 nodes), the most popular solution methods are Branch Exchange (BXC), Concave Branch Elimination (CBE) and Cut Saturation (CS) [2, 12]. Lavia and Manning [10] have proposed perturbation techniques under connectivity and diameter constraint. Moreover, for large computer networks (more than 100 nodes), Kleinrock and Kamoun [9] have elaborated optimal clustering structures for hierarchical topological design, while Chen et al. [1] proposed an extended model and a solution method for network topological design, taking into account the selection of switching node locations.

These methods present two major disadvantages:

- they cannot deal with high degree of reliability (connectivity greater than 2) which is required by the large computer networks;

- they require human intervention for obtaining alternate solutions, by minor modifications on a given solution.

3. A KNOWLEDGE-BASED APPROACH

This approach consists in generating an initial topology well characterized, on which some perturbations are applied by an knowledge-based system in order to obtain a good suboptimal solution, lower-cost topology satisfying all constraints of the specified problem [4, 7, 8]. An inductive learning module is also available for the evaluation of rules already stored in the rule base and the generation of new rules from knowledge contained in the system. In this section, we explain the proposed approach and present the architecture of the system.

3.1 GENERAL ORGANIZATION

From data specified by a user, a good starting topology is first generated. Rules are applied on this topology for providing positive examples (good topologies satisfying all constraints, particularly the delay constraint) and negative examples (good topologies violating the delay constraint). All positive examples determine a set of feasible good topologies, and a solution corresponds to the least cost topology of this set. Furthermore, the generated examples are submitted to an inductive learning module, whose the role is to improve the rules for generating examples. More precisely, this module deals with:

- the detection and correction of rule inconsistencies;
- the elimination of rule redundancies;
- the addition of new knowledge;
- the rule updates;
- etc..

The system is decomposed into four major functional modules, as follows:

- the initial topology generator which produces a starting topology satisfying the K-connectivity constraint;

- the example generator playing the role of an rule-based system or expert system, and using heuristic perturbations for generating positive and negative examples from the starting topology;

- the inductive learning module which receives a set of nondeterministic rules and a collection of representative examples, and improves the rule base; and

- the user interface module which permits interactions between (expert and nonexpert) users and the system, particularly in order to specify data and parameters characterizing the network to design.

3.2 ARCHITECTURE OF THE SYSTEM

Figure 1 gives a detailed representation of the problem-solving system. In order to understand it, we first present some basic definitions, then a summary of used notations and finally the general algorithm.

3.2.1 Basic Definitions

The rules can be deterministic or nondeterministic. Deterministic rules generally express the analytic properties of generated initial topologies. They serve to describe absolute truth contexts, and are consequently accompanied by likelihood factors equal to one. On the other hand, a rule is nondeterministic when it refers to an uncertainty situation, expressed by a likelihood factor less than one. These rules are inspired either by conventional heuristics or experimental methods of machine learning from examples. Obviously, the likelihood factors are nonnegative real numbers not greater than one.

When a starting topology is submitted to the example generator, all applicable rules are applied to it, in order to generate new derived topologies, called examples, which are stored in the knowledge base. This is called a "perturbation cycle". For the first perturbation cycle, the starting topology is generated by the initial topology generator and is consequently called an "initial topology". For the subsequent perturbation cycles, the starting topology is somehow selected among these derived exemples and is renamed a "reference topology". So, for a given design task, it can exist many reference topologies, but only one related initial topology. Similarly, we can define a "learning cycle" as the process allowing to modify the base of nondeterministic rules, on user requests.

3.2.2 Summary of Notations

The meanings of notations used in figure 1 are as follows:

F : an information vector submitted by the user interface module to the initial topology generator; it contains the specifications which are necessary to start the system.

q : a question/answer vector exchanged between the user interface module and the system; according to the nature of the dialogue, the example generator appears as the unit which interprets, formulates and fulfils user requests.

X : an example base acting as input to the inductive learning module, which is accumulated during the life time of the system.

E : an example base accumulated during the solution of the current design problem.

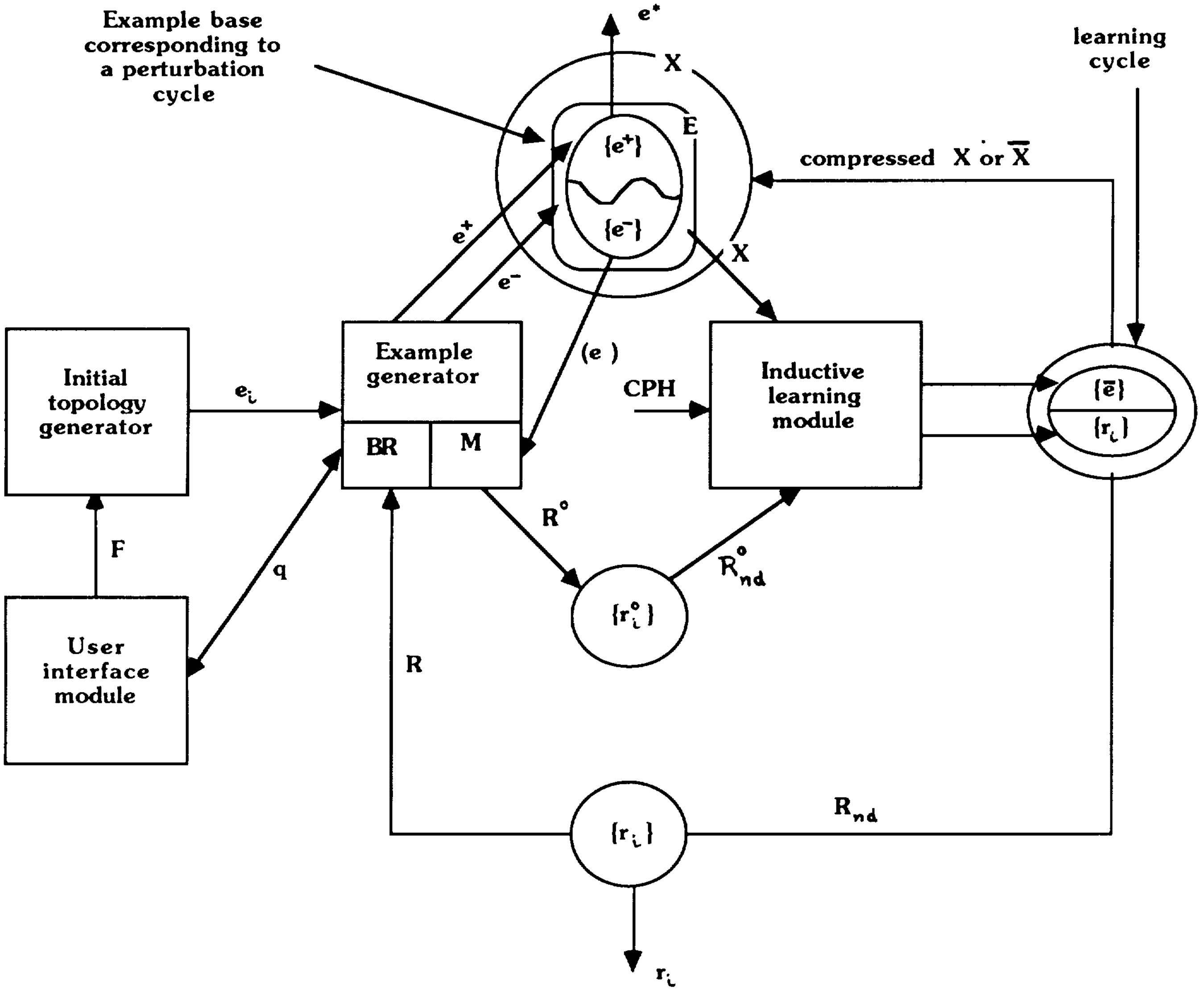


Fig. 1 - Detailed architecture of the system

e_i: an initial topology.

BR: a rule base allowing to generate examples or facts which constitute E.

M : an inference engine, allowing to apply the rules in BR to the examples in E, which are considered as facts.

e^+: a positive example provided by the example generator.

e^-: a negative example provided by the example generator.

e^*: the best feasible solution so far obtained during one or more perturbation cycles already performed.

$\{e^+\}$: a set of positive examples accumulated.

$\{e^-\}$: a set of negative examples accumulated.

(e): the least cost example in E given to the example generator to start a new perturbation cycle.

$\bar{e}$: a representative example selected by the inductive learning module.

CPH: a hypothesis preference criterion, allowing to discriminate plausible assumptions in the learning process.

R^0: a base of initial rules, including both deterministic and nondeterministic rules.

$r_i{}^0$: a nondeterministic initial rule.

$\{r_i^o\}$: a subset of nondeterministic initial rules.

R^o_{nd}: a base of nondeterministic initial rules.

r_i: a nondeterministic rule resulting from a learning cycle.

$\{r_i\}$: a subset of nondeterministic rules accumulated during a learning cycle.

R_{nd}: a base of nondeterministic rules resulting form a learning cycle.

R: a new rule base, obtained by an union of the subset R_d of deterministic rules and the base R_{nd} of nondeterministic rules resulting from a learning cycle.

3.2.3 General Algorithm

The general algorithm is defined by the following steps:

Step 1 : The user interface module transmits to the initial topology generator the information vector F specifying the context of the design.

Step 2 : The initial topology generator produces a starting example or initial topology e_i satisfying the specification vector F. The example base E is empty.

Step 3 : The example generator applies the rule base to the starting topology to generate positive examples e^+ and negative examples e^- satisfying the specification vector F.

All generated examples are included in E.

Step 4 : At the end of a perturbation cycle, the system proposes the best feasible feasible solution obtained, that is,

$$e^* = \min_D \{e^+\}$$

Step 5 : The user interface module, via the vector q, possibly asks for explanations about the proposed solution, generation of a new solution, learning new rules, and so on.

Step 6 : If a new solution is required, the example generator applies again the rule base to a new reference topology or example (e), which is the least cost example in E, that is, go back to step 3.

Step 7 : If a learning cycle is required, the inductive learning module receives the example base X, the hypothesis preference criterion CPH and the base of nondeterministic rules R^0_{nd}, induces new nondeterministic rules $\{r_i\}$, and constructs a new abstract and compressed representation of X called $\overline{X}$.

Step 8 : At the end of a learning cycle, the example base X is updated by the assignment X := X U X, the base of nondeterministic rules R^0_{nd} is then replaced by the new base R_{nd}. The result of that is a new rule base defined by the assignment

$R := R_d \cup R_{nd}$.

Step 9 : The user interface module, via the vector q, possibly asks to display the new induced rules, to modify the rule base, to submit a new vector q, to stop the running of the system.

Step 10 : Stop.

4. KNOWLEDGE ORGANIZATION

The example generator which is represented in figure 1 acts as a knowledge-based module. It essentially consists of a rule base and an inference engine. In this section, we explain the operating of the example generator and deal with the rule base organization.

4.1 THE EXAMPLE GENERATOR

When the initial topology generator provides a particular initial topology considered as a starting topology (t^0,c^0) from the problem specifications, the example generator receives this topology and applies its knowledge base to transform (t^0,c^0) into (t^i,c^i), i= 1,2,...

The initial topology is characterized by the following attributes: a number of nodes, a number of links, a link flow vector, a link capacity vector, a link utilization vector, a degree of connectivity, an average delay, a total communication design and other secondary attributes mainly used by the learning process. These are the features of the concept of example. Moreover, an example whose the average delay is greater than the maximum allowed delay is classified as a "negative example"; otherwise it is a "positive example". So, the initial topology must be considered as the first generated example, which can be positive or negative.

The example generator is essentially composed of two parts: the knowledge base which can be subdivided into an example base and a rule base, and the inference engine playing the role of a control program.

The example base is further divided into a long-term example base and a short-term one. The long-term example base, which is empty when the system is freshly installed, contains highly discriminating examples that have been discovered during the system lifetime. It requires a highly abstract, compressed and efficient representation of examples for the purposes of machine learning. The short-term example base is composed of the examples generated during the solution process of a specific problem. The first example introduced into the example base is the initial topology.

The rule base contains perturbation rules which can be applied to the starting topology (t^0,C^0), in order to obtain potentially better new examples, that is, K-connected topologies improving the cost or the average delay in comparison with the starting topology. The inference engine selects the appropriate rules R_i, $i=1,2,\ldots$, and applies them to (t^0,C^0). Each applied rule R_i generates an example e_i which is stored in the short-term example base. When all the rules were considered with regard to that starting topology, a perturbation cycle had then completed. At the end of a successful cycle, the expert module produces one solution, corresponding to the lowest cost positive example stored in the short-term example base. A new perturbation cycle can be performed by choosing an appropriate element in the short-term example base as new starting topology. This is the reference topology related to this perturbation cycle.

4.2 RULE BASE ORGANIZATION

For the efficient operations of the expert module, the set of rules have been partitionned in seven categories:

- rules defining positive and negative examples;
- rules for perturbation selection;
- capacity modification rules;
- link addition rules;
- link deletion rules;
- link substitution rules; and
- connectivity preserving rules.

4.2.1 Rules Defining Positive and Negative Examples

These rules allow us to group together, as positive examples, topologies which satisfy the entire set of performance constraints, and as negative examples those which do not satisfy the delay constraint while respecting all other constraints. There are two such rules, which are deterministic:

```
If  1) a topology satisfies the
       connectivit constraint
    2) the mean delay of the
       topology is greater than
       the maximum allowed
Then classify the topology as a
     negative example
```

```
If  1) a topology satisfies
       the connectivity
       constraint
    2) the mean delay of the
       topology is not greater
       than the maximum allowed
Then classify the topology as a
     positive example
```

4.2.2 Perturbation Selecting Rules

The rules of this category need be considered in the perturbation process, in order to reduce the costs and/or average delays of topologies by applying appropriate perturbations. Perturbations essentially consist in modifying link capacities, and in adding, deleting or replacing links. Those rules can be either deterministic or nondeterministic. Here is an example:

```
If   1) the topology is initial
     2) the topology contains
        more than 5 nodes
     3) the degree of
        connectivity is greater
        than 4 but less than
        (n - 1)
Then the recommended
     perturbation is a link
     substitution (C.V.: 0.70)
```

4.3.3 Capacity Modification Rules

The capacity modification rules define a context in which the cost or average delay of a topology can be reduced if link capacities are modified. Those rules are nondeterministic. One example of such rules is the following:

```
If  1) the average delay of the
       topology is not greater
       than the maximum allowed
    2) the envisioned
       perturbation is a
       capacity modification
Then a downward capacity
     adjustment can reduce the
     cost (C.V.: 0.80)
```

4.3.4 Link Addition, Deletion and Substitution Rules

Those three categories of rules define a context in which links addition, deletion or substitution can reduce the total cost. Those rules can be deterministic or nondeterministic. Here is an example:

```
If  1) the topology is initial
    2) the topology contains at
       least 5 nodes
    3) its degree of
connectivity
       is equal to 2
    4) the envisioned
       perturbation is a link
       deletion
Then   delete the (n - 3) links
     which are connected to
     nodes of degree greater
     than 2
```

4.3.5 Connectivity Preserving Rules

Connectivity preserving rules mainly aim at necessary conditions to satisfy the connectivity constraint while applying link deletion rules. They are based on the set of propositions expressing the analytical properties of initial topologies. The following is an example:

If 1) the related initial topology contains more than 4 nodes
2) the related initial topology has a degree of connectivity equal to 2
Then at most (n- 3) links can be deleted from the reference topology to obtain a derived topology.

5. Inductive Learning

Inductive learning is defined as the acquisition of knowledge by means inductive inferences which are effectuated from facts provided by a teacher or an environment (Mitchell, Carbonell and Michalski 1986). The related module aims at improving the rule base in order to achieve more refined inferences. In this section, we first formulate our inductive learning problem, then we present an appropriate algorithm.

5.1 LEARNING CHARACTERIZATION

The implemented learning is incremental, with partial-memory of past examples [15]. It can be formulated in the following terms:

Given:

- a nondeterministic rule base, R_{nd}
- an example base, E
- with each rule r_i of R_{nd} is associated a candidate hypothesis space H
- an hypothesis preference criterion CPH which permits to select amongst a set of plausible hypotheses.

Objective:

- Find - by generalization, specialization or reformulation- a new nondeterministic rule base R_{nd} such as the description $R = R_d \cup R_{nd}$ consistently covers the near total of good examples stored in E.

5.2 Learning Algorithm

The proposed inductive learning algorithm is defined by the following steps:

Step 1 : Receive the set R^0_{nd} of nondeterministic initial rules and do $R_{nd} := R_{nd}$;

Step 2 : Receive from the example base E an example e, then build the subset $R_{nd}(e)$ of nondeterministic rules which has generated e, where $e \varepsilon E$ and $R_{nd}(e) \subset R_{nd}$;

Step 3 : If e is a positive example, then the nondeterministic rules $R_{nd}(e)$ which have generated it are checked:

. update the likelihood factors, if necessary;
. make a list of discriminating properties of e which could imply the generality of rules in the subset $R_{nd}(e)$;
. generalize, if necessary, the rules of R_{nd} taking into account the related hypothesis spaces; if there is conflict in the

selection of hypotheses, use the given hypothesis preference criterion to solve it;

Step 4 : If e is a negative example, then at least one of nondeterministic rules which have generated it is not confirmed:

. update the likelihood factors;

. make a list of involved discriminating properties of e;

. specialize or reformulate the rules of $R_{nd}(e)$ taking into account the related hypothesis spaces; if there is conflict in the selection of hypotheses, use the given hypothesis criterion to solve it;

Step 5 : If at least one example of the rule base E is not yet considered, then go to step 2;

Step 6 : Stop.

6. Computational Experience and Concluding Remarks

In order to evaluate the efficiency of our method, now implemented on a typical IBM PC AT, we have considered a set of fifty network problems, randomly generated, which have been also solved by the cut saturation method. For a convenient comparison with the cut saturation method, our experience is based on the following choices:

- the number of nodes is always kept equal to 25;
- traffic is constant between each pair of nodes;
- the degree of connectivity is always equal to 2;
- the maximum delay is T_{max} = 200 msec;
- the average size of data packets is equal to 1000 bits/packet.

For a given problem, a solution is characterized by a topological configuration t, a capacity vector C, a flow vector f, an average delay T, a transmission links cost D, and CPU time. In 80 % of cases, solution provided by our method gives a lower cost than the cut saturation solution. Furthermore, in 90 % of cases, the CPU time required to provide a solution is lower in the case of SIDROGT than cut saturation.

In this paper, we have presented an artificial intelligence approach for solving the network design problem. The heart of this approach is constituted by an expert module which receives an starting topology and operates on it local transformations by means heuristic perturbations. An inductive learning module is used for improving the efficiency of those transformations.

Solution provided by such a system is obviously suboptimal. But, it is made up by an computationally efficient and flexible process which allows to attempt a new solution by initiating a new perturbation cycle, or to improve the rule base by initiating a new learning cycle. Furthermore, another advantage of that system is the high degree of connectivity which it permits. The initial topology generator provides topologies which are 1, 2,..., (n-1) connected (where n denotes the number of nodes), satisfying by the way the reliability constraint. That is truly innovative in comparison with the other methods, which are limited to the 2-connectivity and generally start with an unrefined starting topology. The degree of connectivity is preserved by both the knowledge-based module and the inductive learning module. So, it is not necessary to run a time-consuming connectivity-restoring algorithm. For those reasons, such a system is suitable for designing large scale computer networks, where a high level of reliability is required.

References

[1] K.J. Chen, J.F. Stach, and T.H. Wu, "A New Method for Topological Design in Large, Traffic Laden Packet Switched Networks,"Proc. Ninth Data Communications Symposium, ACM SIGCOMM Computer Communication Review, Vol.15, No.4, 1985, pp.115-121.

[2] M. Gerla and L.Kleinrock, "On the Topological Design of Distributed Computer Networks," IEEE Trans. on Communications, Vol.Com-25, No.1, 1977, pp.48-60.

[3] R.R. Boorstyn and H. Frank, "Large-Scale Network Topological Optimization," IEEE Trans. on Communications, Com-25, No.1, 1977, pp.29-47.

[4] E. Hausen-Tropper, H.H. Hoang, S.Pierre, "Approche Système Expert pour la Conception de Réseaux Téléinformatiques de Grande Taille", Neuvièmes Journées Internationales sur les Systèmes Experts et leurs Applications, Avignon, France, in Intelligence Artificielle et Télécommunications, 1989, pp. 134-148.

[5] R.M. Karp, "Combinatorics, Complexity, and Randomness," CACM, Vol.29, No.2, 1986, pp.97-109.

[6] D.S. Johnson, J.K. Lenstra and A.H.G. Rinnooy, "The Complexity of the Network Design Problem," Networks, Vol.8, No.4, 1978, pp.279-285.

[7] S. Pierre and E. Hausen-Tropper, "Design Topologique de Réseaux Téléinformatiques de Grande Taille: une Nouvelle Approche," IEEE Montech'86, Montréal, 1986, pp.370-372.

[8] S. Pierre, H.H. Hoang, E. Tropper, "An Expert Systems Application in Computer Network Topological Design," IASTED International Conference on Expert Systems, Theory & Applications, Zurich, Switzerland, 1989, pp. 139-142.

[9] L. Kleinrock and F. Kamoun, "Optimal Clustering Structures for Hierarchical Topological Design of Large Computer Networks," Networks, Vol.10, 1980, pp.221-248.

[10] A. Lavia and E.G. Manning, "Perturbations Techniques for Topological Optimization of Computer Networks," Proceedings of Fourth Data Communications Symposium, 7-9 oct. 1975, Quebec City, pp.4-16, 4-24.

[11] E.A. Sykes and C.C. White, "Specifications of a Knowledge System for Packet-Switched Data Network Topological Design," Expert Systems in Government Symposium, McLean, VA, Oct. 1985, pp. 102-110.

[12] M. GERLA, H. FRANK, W. CHOU, and J. ECKL, "A Cut Saturation Algorithm for Topological Design of Packet-Switched Communication Networks," Proc. Nat. Telecom. Conf., 1974, pp. 1074-1085.

[13] H.H HOANG, "Topological Optimization of Networks: A Nonlinear Mixed Integer Model Employing Generalized Benders Decomposition," IEEE Transactions on Automatic Control, Vol. Ac-27, No. 1, 1982, pp. 165-169.

[14] R. KRONZ, S. LEE, and M. SUN, "Practical Design Tools for Large Packet-Switched Networks," Proc. IEEE INFOCOM, 1983, pp. 591-599.

[15] T.M. MITCHELL, J.G. CARBONELL, and R.S. MICHALSKI, MachineLearning: A Guide to Current Research, Kluwer Academic Publishers, 1986.

EVALUATION OF A PROPOSED EXPERT SYSTEM DEVELOPMENT METHODOLOGY: TWO CASE STUDIES

Lewey Gilstrap
Computer Sciences Corporation
4600 Powder Mill Road
Beltsville, MD 20705

ABSTRACT

Two expert system development projects were studied to evaluate a proposed Expert Systems Development Methodology (ESDM). The ESDM was developed for use at Goddard Space Flight Center (GSFC) to provide guidance to managers and technical personnel and serve as a standard in the development of expert systems. It was agreed that the proposed ESDM must be evaluated before it could be adopted; therefore a study was planned for its evaluation. This detailed study is now underway. Before the study began, however, two ongoing projects were selected for a retrospective evaluation. They were the Ranging Equipment Diagnostic Expert System (REDEX) and the Backup Control Mode Analysis and Utility System (BCAUS). Both projects were approximately 1 year into development. Interviews of project personnel were conducted, and the resulting data was used to prepare the retrospective evaluation. Decision models of the two projects were constructed and used to evaluate the completeness and accuracy of key provisions of ESDM. A major conclusion reached from these case studies is that suitability and risk analysis should be required for all AI projects, large and small. Further, the objectives of each stage of development during a project should be selected to reduce the next largest area of risk or uncertainty on the project.

INTRODUCTION

The Expert Systems Development Methodology (ESDM) is intended to be applied to the development of expert systems at the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC). The methodology is based on a survey of existing methodologies, experience in developing a number of expert systems at GSFC, and an analysis of the expert system life cycle. Dr. Barry W. Boehm introduced a risk-driven methodology for conventional systems development in his spiral model for software development (Boehm, 1988). ESDM, while independently generated, is also a risk-driven methodology that can be represented by a spiral model with the focus on knowledge acquisition as opposed to product development. Figure 1 shows the spiral model of ESDM.

Risks are inherent in all system development projects, but they are greater in ES development because of the uncertainties associated with modeling human expert decision processes. At the outset of the development of an expert system, it is not known whether an expert's decision processes are cognitive processes that can be modeled by ES techniques. Some human decisions are made on the basis of intuition or skills, which usually cannot be modeled using ES techniques. Intuitive processes and skills can often be modeled using other techniques, such as neural networks, but ESDM does not address these. Even after it has been determined that an expert's decision processes can be modeled, there remain developmental risks because of uncertainties about the robustness and performance that can be obtained from the expert system.

ESDM was developed as a tool for both project managers and developers of expert systems in the NASA environment. It focuses on the knowledge acquisition task, rather than on product development. Key features and recommendations of ESDM include:

- Decomposition of an ES development project into stages. In each stage, work is directed toward the acquisition of the key knowledge needed to reduce the most immediate or highest level risk of the project.
- Explicit identification of the objectives of each stage of work prior to its initiation and testing to verify that the objectives have been met.
- Well-defined criteria for stopping ES development. Once the functional requirements of the proposed system have been identified, ESDM recommends dropping the ES approach and continuing the project along the lines of conventional system development. ESDM also recommends stopping the ES project if the expert's decision processes are not suitable for ES modeling or if an

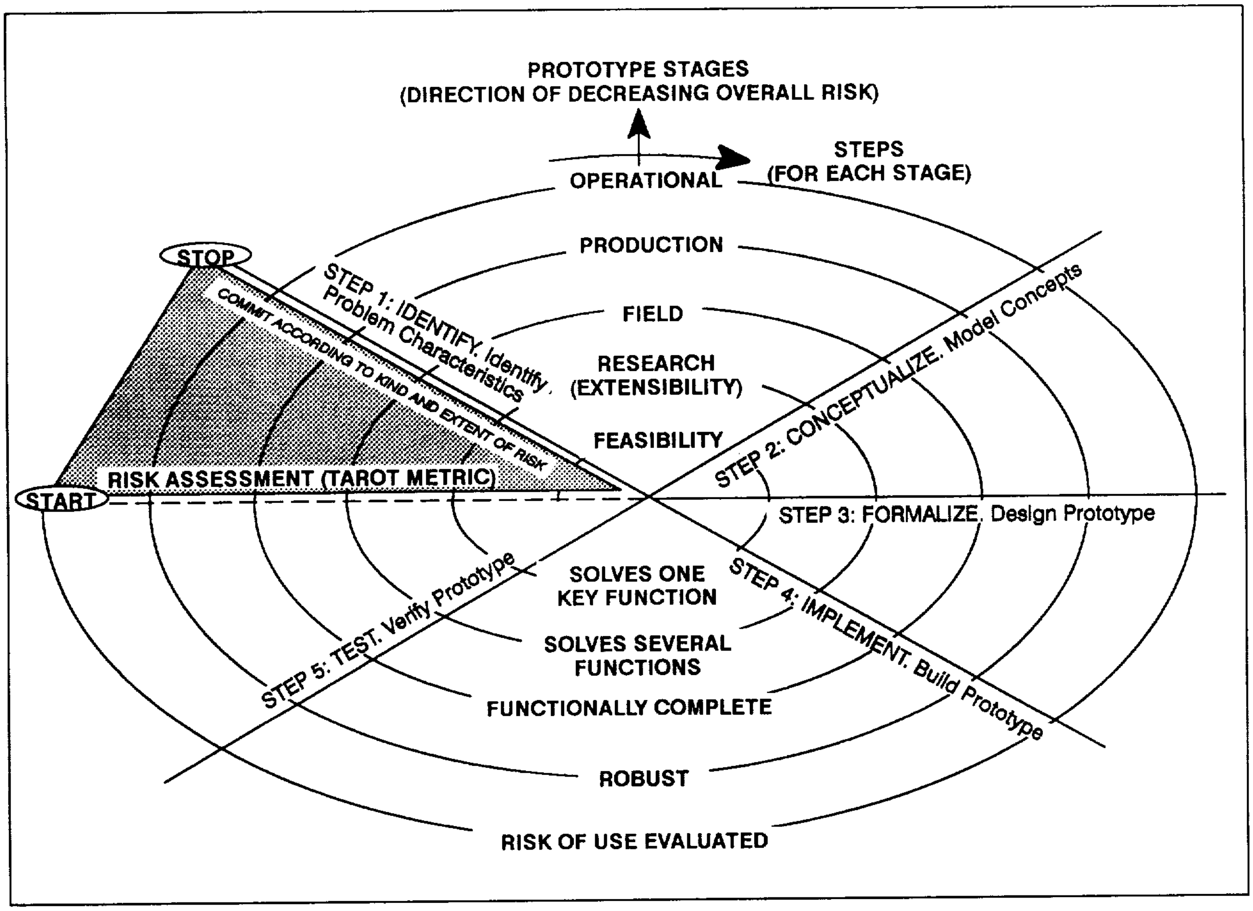

PROTOTYPE STAGES
(DIRECTION OF DECREASING OVERALL RISK)
STEPS
(FOR EACH STAGE)
OPERATIONAL
PRODUCTION
FIELD
RESEARCH
(EXTENSIBILITY)
FEASIBILITY
STOP
START
RISK ASSESSMENT (TAROT METRIC)
COMMIT ACCORDING TO KIND AND EXTENT OF RISK
STEP 1: IDENTIFY. Identify Problem Characteristics
STEP 2: CONCEPTUALIZE. Model Concepts
STEP 3: FORMALIZE. Design Prototype
STEP 4: IMPLEMENT. Build Prototype
STEP 5: TEST. Verify Prototype
SOLVES ONE
KEY FUNCTION
SOLVES SEVERAL
FUNCTIONS
FUNCTIONALLY COMPLETE
ROBUST
RISK OF USE EVALUATED

Figure 1. Spiral Model of ESDM

algorithm is discovered that performs the decision process satisfactorily.

- ESDM provides guidance for the kinds of special documentation needed for ES projects and the management information that should be collected for administrative reporting.
- ESDM recommends using of quantitative methods for assessing risk where possible and provides a tool, the Test for Application of Risk-Oriented Technology (TAROT), to assist in this evaluation.

The ESDM project has produced a user's guide (CSCa, 1988), a policy document (CSCb, 1988), and a reference manual (CSCc, 1988) along with training materials. ESDM has been proposed for use on all GSFC expert system projects. Because a proposed methodology must be evaluated before its adoption, however, a framework for the evaluation was also developed (CSC, 1989). The framework recommended selecting an expert system development project and using it as a pilot to evaluate the features of the ESDM before its adoption as a standard. The project would be followed from beginning to end and would collect data on ESDM effectiveness.

Before undertaking a full-scale pilot study, two ongoing projects were selected for a retrospective evaluation of ESDM. The Ranging Equipment Diagnostic Expert System (REDEX) and the Backup Control Mode Analysis and Utility Systems (BCAUS) were the two projects selected. Data on the two projects was collected by interviewing project personnel. Decision models of the two projects were also constructed and used to evaluate the completeness and accuracy of ESDM in accordance with the general provisions of the framework for evaluation.

This paper presents a summary of the findings made on these two case studies. The case studies include a description of the two projects, the key decisions made on the projects, and conclusions reached about the methodology.

THE REDEX SYSTEM

REDEX is an advanced prototype expert system that diagnoses hardware failures in the ranging equipment (RE) at NASA's Ground Network tracking stations (Luczak, 1989). REDEX is intended for use by RE technicians in identifying faulty circuit cards or modules that must be replaced. The system has a highly graphical user interface that uses color block and layout diagrams to illustrate fault locations. Figure 2 shows the environment for REDEX.

The REDEX project was initiated by the Telecommunication Systems Branch (Code 531) at GSFC as a task assignment. There were two persons assigned to the project initially, but the level of effort has averaged less than two full-time persons.

No formal risk or suitability analysis of the project was performed. The use of an expert system as a diagnostic aid for the RE was considered feasible because the RE had been designed with a large number of built-in test points. It was expected that these test points would greatly facilitate the automation of fault diagnosis, and the task of REDEX was to speed up the identification process.

Development staff personnel were generally familiar with the provisions of ESDM. On their own initiative, they selected ESDM features that they believed would assist them in the development of REDEX and used them in the project. The selected features were:

- The use of a staged development
- The decomposition of stages into steps
- The use of risk analysis to guide the selection of objectives for stages

The stages of work on REDEX followed ESDM recommendations closely for addressing successively more complex objectives. Five stages of work were defined, each addressing more complex issues. The following summarizes these five stages:

1. Feasibility of implementing one diagnostic rule and accomplishing diagnosis with this rule
2. Feasibility of extending the feasibility prototype to include all relevant rules on the selected hardware host (IBM PC-AT)
3. Feasibility of implementing one graphics screen on the selected host
4. Feasibility of extending the graphics system to include all required graphics
5. The capability of the system to be fielded (field prototype), including handling all necessary communications with the equipment

REDEX is implemented in Prolog on an IBM PC AT-compatible workstation. A semantic network knowledge representation technique was used to model the design structure of the RE. A catalog of generic troubleshooting rules was compiled to represent heuristics that are applied in diagnosis. Specific troubleshooting rules unique to the RE were also added. Over 50 generic and 250 specific rules were developed. A hypertext-like scheme is used to allow the user to navigate through the diagrams and tables. Over 50 graphic and tabular displays have been implemented.

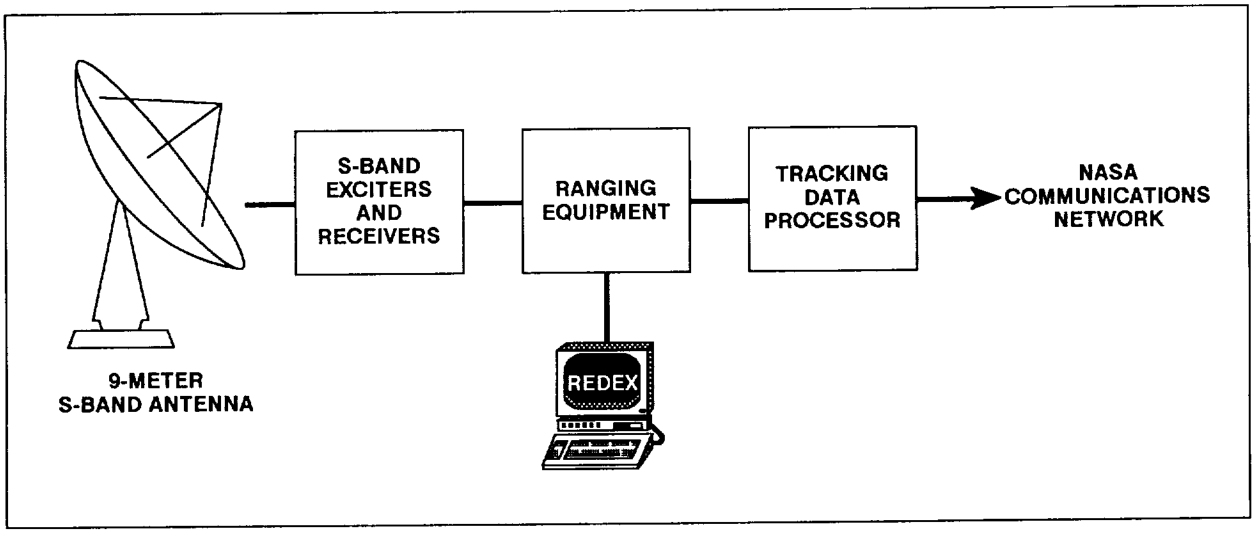


Figure 2. NASA Ground Network Tracking Station

The client identified the use of IBM PC-AT hardware as desirable at the outset of the project. The choice of computer language or shell development system was left to project personnel, but an early identification of the language was requested for budgetary estimation. At project initiation, it was not clear what knowledge representation scheme might be most appropriate for the knowledge yet to be acquired, and project personnel began addressing this problem immediately. After investigation, they determined that a rules representation would be appropriate. Generic and specific rules (in a pseudocode form) for the system were manually compiled.

From the somewhat limited choices available for the implementation of knowledge-based systems on an IBM PC-AT, project personnel concluded, after further investigation of six languages and shell development systems, that Prolog appeared to be suitable for prototyping the rules. A feasibility prototype was then developed to test whether Prolog was suitable for the system prototypes. (Prolog's suitability for the operational system was not determined at this time; however, its suitability in this regard has since been proved.) At the completion of the second prototype stage, called the research stage in ESDM, the functional requirements for REDEX were validated. These requirements were then documented and issued in a functional requirements document.

The fifth (and current) prototype stage of REDEX addresses those risks associated with using the system in the field. In this prototype, the uncertainties are concerned with the communications between the RE and REDEX. This system is currently being evaluated in a communications emulation testbed and will be connected to the RE after evaluation.

THE BCAUS SYSTEM

BCAUS is an expert system designed to assist flight operations personnel in diagnosing the cause of a Gamma Ray Observatory (GRO) spacecraft autonomous mode transition (Bush, 1989). The GRO spacecraft was designed with onboard capability to safe itself autonomously, transitioning from a primary operating mode to a backup control (safing) mode in the event of certain error conditions in the attitude control and determination (ACAD) subsystem.

Flight operations personnel need to understand what error condition trigger the onboard computer (OBC) to order the mode transition and why that error condition occurred so that they may take the proper corrective action. The OBC was not designed, however, to provide the triggering information or the diagnostic information to the operator.

Input information to BCAUS will be provided by telemetry data from GRO and by user input. Output from BCAUS will be provided only to the diagnostician. There is no output back to the spacecraft. Figure 3 shows a diagram of the information flows in BCAUS.

GSFC also initiated the BCAUS project by issuing a task assignment. Two persons were assigned to the project. No formal risk or suitability analysis of BCAUS was performed. Task personnel had knowledge of the risk areas in expert system development and used this information to guide the development process. The primary area of risk for BCAUS was in the knowledge acquisition process. Four sources of expertise were identified and were initially considered adequate for the development task. These four sources were (1) documentation, (2) GSFC spacecraft design experts, (3) GSFC flight operations experts, and (4) TRW personnel associated with the design of the relevant GRO subsystems. However, project personnel found that the knowledge acquisition task for this system was more difficult than initially thought and that the initial evaluation of risk had to be modified. The project goals have therefore shifted from providing an operational system to a system in which the knowledge base is easily modified and updated on the basis of actual experience. In brief, the goal has shifted from providing an initially operational system to an adaptive system with an initial base of knowledge that can be upgraded as expertise is acquired.

The difficulty in the knowledge acquisition task for the BCAUS project is that the expertise needed to diagnose GRO mode transitions has not yet been acquired by humans. There was no training course available for GRO fault diagnosis as there was for REDEX. The existence of a training course means that the diagnostic knowledge has been compiled, thus implying a lower risk of system development. However, even the designers of the GRO subsystems had not yet acquired or compiled all the information necessary for mode transition analysis, and this fact was not known at the outset of the project. The relative inaccessibility of the TRW design engineers because of their location on the west coast and their limited availability for consultation made it difficult for project personnel to elicit any available information. When the full difficulty of knowledge acquisition became known, a reevaluation and reorientation of project goals and objectives became necessary. The complexity of the knowledge acquisition task on the BCAUS project perhaps doubled the time required to reach a feasibility prototype. This situation constrained the design of the operational system in ways

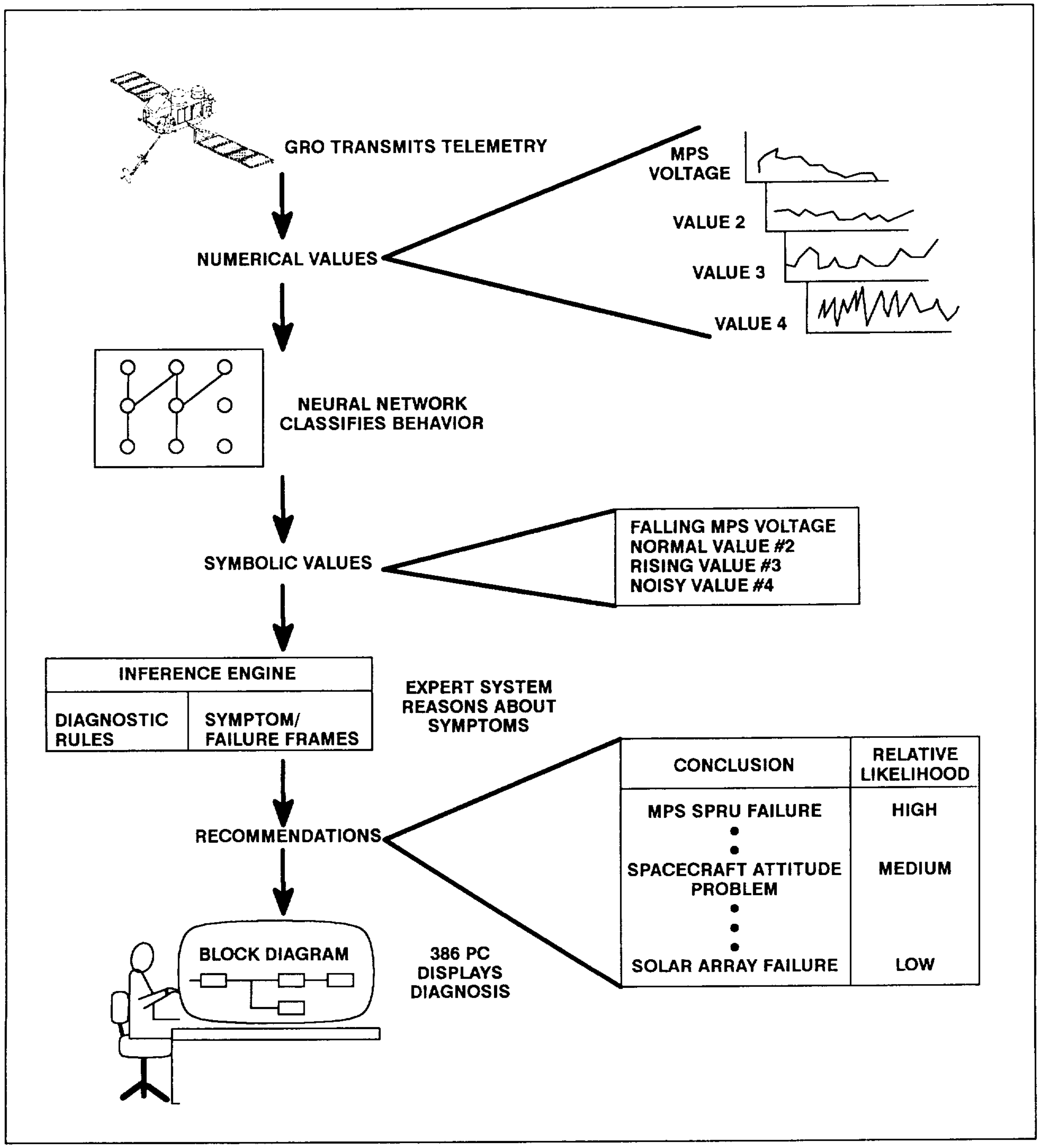


Figure 3. The BCAUS System

that were not and probably could not have been, determined at the outset.

The first prototype system, a feasibility prototype, was implemented on a PC-386 class machine using the KES hypothesize-and-test (HT) inference engine developed by Software Architecture and Engineering, Inc. Basic structural knowledge of the system elements was loaded on the machine in three weeks by two persons. No rules were needed because of the built-in diagnostic feature of KES HT. When KES HT was selected initially, there were some known limitations on its capabilities. Following its use on the project, it became clear that the limitations were too restrictive, especially in the area of explanatory power. The second prototyping system selected was ART-IM, an expert system development shell produced by Inference Corporation, which can run on both the 386 machine and a Silicon Graphics Iris 4D/20. The Iris has considerably more power for graphics than the 386 machine.

The BCAUS system will also have a neural network front end to provide trending data on input telemetry signals. Project personnel determined that it might be possible to implement trending analysis using ART-IM rules, but at the expense of making the system much larger and more complex than desirable. The software product, Neural Works Professional II, from Neural Ware, Inc., was selected to implement the trending analysis.

The BCAUS system's graphics interface shows relevant subsystems in the form of functional block diagrams, similar to those implemented in REDEX, with highlighted potential problem areas. A hierarchical traverse is planned for navigation of the diagrams and causal graphs.

As was the case with REDEX, there was a strong tendency to follow the methodology used for conventional software systems, and the hardware and software selections were set very early in the project. The deadlines for hardware and software selection were met only through very concentrated effort on the part of the project's development staff.

PROJECT KEY DECISIONS

In the evaluation process, ESDM was modeled as a sequence of key decisions plus subsidiary decisions. Key decisions are identified on the basis of their possible impacts on project cost and schedule. The key decisions of ESDM are:

- Start. The decision that the project is suitable for implementation by an expert system is based on a formal suitability analysis in ESDM.
- Knowledge-oriented approach. Is current knowledge about the problem sufficient to permit preparation of specifications for the system now? If not, then a knowledge-oriented approach is indicated, that is, the decision is made to acquire the missing information first.
- Staffing. ESDM recommends a knowledge engineer, AI programmers, system programmers, and domain experts for expert system projects.
- Staging. ESDM recommends dividing a project into successive stages based on the degree of uncertainty about how to accomplish the function or service.
- Steps within stages. ESDM recommends following five steps within each stage. These steps focus on identifying the knowledge to be acquired within the stage, on acquiring this knowledge, and on verifying the correctness of the acquired knowledge.
- Explicit risk evaluation. ESDM recommends that all risk evaluation be explicit, that is, that each area of risk on the project be documented and assessed. ESDM also provides a formal tool, the TAROT metric, to assist in estimating of risks.
- Stop-rule. This decision is based on acquiring sufficient information to prepare meaningful and realizable specifications. At this point, ESDM recommends continuing the project as a normal software development project following the conventional software development life cycle.

There are also technical and managerial decisions of lesser importance that have some bearing on project schedule and costs:

- Reporting (frequency, type, content)
- Hardware and software tools selection and timing
- Use of automated knowledge tools
- Need for graphics and interfaces

The method used to evaluate ESDM was, first, rating how closely the circumstances and decisions of the two projects matched the provisions of ESDM and, second, assessing the worth of the provisions based on the experience gained on the projects.

The start decision on both REDEX and BCAUS, that is, the decision to use an expert system or knowledge-based technology to develop a system was made by GSFC personnel, not by the development project personnel. There is no information on whether any formal analysis of suitability was made by GSFC personnel. In retrospect, it is clear that both projects were, in fact, suitable. By now, the usefulness of expert systems for fault diagnosis has been well established; this fact can be considered generally well known in the computer field.

The ESDM provision that recommends a suitability analysis for each new project should be amended to take into account current practices, informal standards, and common knowledge among practitioners in the computer field. The study concluded that a more formal analysis of suitability should still be performed for any system that does not fall into one of the familiar categories of expert system applications. The decision regarding a formal suitability analysis on any new project is a judgment call. Nevertheless, ESDM must continued to provide the guidelines and procedures for cases requiring a suitability analysis.

It was apparent even from a casual analysis of both REDEX and BCAUS that it was impossible to prepare specifications at the outset for either project and that a knowledge acquisition process would be required. What is important, however, is that knowledge acquisition procedures are required and that identifying the missing pieces of information is necessary in order to develop the systems. The identification of this information was carried out on both projects.

Both the REDEX and BECAUS projects were staffed with experienced AI professionals. ESDM guidelines call for both knowledge engineers and AI programmers. Because of the small size of the projects, however, it was necessary for project personnel to function both as knowledge engineers and as AI programmers. Also, project personnel assumed some of the functions of domain experts ESDM should be modified to take the special requirements of small projects into account, but the need for experienced and competent staff personnel becomes even more acute for these smaller projects. Managers should remain aware of the staffing requirement differences in small and large projects.

Both REDEX and BCAUS were decomposed into successive stages of work. ESDM recommends defining stages in terms of risk and addressing areas of highest risk first. While there was no conscious decision to follow ESDM provisions on selecting stages, REDEX personnel nevertheless followed the feasibility, research, and field stages quite faithfully. REDEX also decomposed the planned system into three subsystems (functional, user interface, and communications interface) and followed the' risk-reduction sequencing in each subsystem. Staging was also followed on BCAUS. The first year of work on BCAUS was considered to be the feasibility stage.

ESDM defines the *research stage* as that stage of work that establishes that one or a small set of rules can be implemented. The issue to be addressed then is whether enough of the required rules can be implemented to make the system practical. A better name for this stage should reflect the intent of the stage, that is, determining how far the feasibility prototype can be extended. The name, *extensibility stage*, has been suggested as a replacement.

Both projects followed some natural sequence of work within the stages that was similar to the steps described in ESDM. In fact, the steps within the stages recommended in ESDM are a paraphrase of the scientific method, which is the model for knowledge acquisition or discovery processes.

There was no formal analysis of risks made on either REDEX or BCAUS; however, both development teams reported being acutely aware of the risks associated with different areas of their projects at all times and stated that their work was governed by this awareness. This awareness of risk characterizes the experience of the development teams. Less experienced personnel might not have the opportunity to put together workable and useful systems.

On small projects, there is less need for formal analysis of risk. The lack of a formal analysis on small projects should not be a concern to managers, as long as the staff is aware of risks and is guided by their consideration. On large projects, the use of a formal risk analysis is still recommended. ESDM provisions are being modified to take the size of the project into account.

There are no plans to transfer REDEX or BCAUS to a conventional development cycle after preparation of system requirements. On small NASA projects, the personnel who began the project will typically carry on the development even after requirements have been specified and risks reduced to an acceptable levels. Transfer to a conventional life cycle with a new development team, which was recommended in ESDM for large projects, will probably be the exception, rather than the rule, for most small projects.

The documentation prepared on the two projects tended to follow the requirements for conventional software development. Although it is impossible to draw conclusions about ESDM provisions for documentation, project personnel felt that the knowledge acquisition process was not adequately documented by the normal system development documents, thus lending support to the ESDM provisions. Some adjustment for the number of documents recommended by ESDM should be made on the basis of project size.

Similar to the typical requirements associated with conventional software engineering, the hardware and software tools to be used on a project must be specified at the outset or as early as possible. Based on the two retrospective studies, the conclusion is that

ESDM provisions are the least risky, that is, delaying the selection of hardware and software tools until after identification of the knowledge structure.

Because of the experiences gained by project personnel on REDEX and BCAUS, they suggested the possibility of using automated tools for logging and managing lists of rules or other knowledge structures. Since the final evaluation is incomplete at this time, there has been no change in ESDM regarding recommendations on the use of automated tools.

In addition to the two projects described in this study, there was a review of a number of other expert systems prepared for NASA. Nearly all of them made use of color graphics interfaces for presentation of information to the users. This fact has some implications for the selection of tools for the development of expert systems and the selection of personnel to work on expert systems. A possible modification to ESDM will point this out and provide guidelines for tool and personnel selection.

REDEX makes use of input data from equipment monitoring points and BCAUS has telemetry inputs from the GRO spacecraft. Many NASA expert systems have sources of input information other than the human. Also, many make use of other techniques than logic programming, such as:

- Neural networks
- Procedural code
- Operating system calls

The staffing requirements provisions of ESDM that address only knowledge engineering and AI programming should be modified to take into account the possible needs for systems programming, neural net programming, and familiarity with telemetry and communications.

CONCLUSIONS

The two projects surveyed match the model of the expert system development life cycle so closely that the experience gained on these projects provides valuable information for ESDM evaluation. The two projects are quite different in detail and dynamics, and they differ from the expected large-size project envisioned by ESDM. The experience of these projects is useful primarily in providing ESDM with extensions to cover the cases of small-size projects.

General conclusions reached from the retrospective study of the two projects include:

- Confirmation of the need for a methodology. The standard systems development methodology matches the life cycle of expert systems poorly. The need for a methodology better suited to the special requirements of expert systems is supported by project experience.
- Support for the use of a risk-based approach. Both project teams reported that they were aware of risks in development and organized their projects to address these risks. ESDM formalizes this practice in ES development.
- The decomposition of projects into successive stages. Both projects broke the work down into successive stages to make the overall task more manageable.
- Requirements as an overall goal. Both projects produced requirements documents at the conclusion of an extensive knowledge acquisition phase in accordance with the recommendations of ESDM.

Based on the evaluations provided by the two projects, REDEX and BCAUS, it was possible to reach some specific conclusions about the details of ESDM and the framework to be used for its evaluation on the two pilot projects.

- ESDM currently requires a formal suitability analysis for all projects. Findings suggest that this requirement should be relaxed for small projects.
- ESDM should be modified to describe the differences between small and large projects. In particular, some of the formal documents required for projects are unnecessary for small projects and may impose an unnecessary burden.
- The name of the research stage of the ESDM life cycle should be changed to the extensibility stage.
- The use of the TAROT metric (or other formal tool) for evaluation of risk and the suitability of candidate projects for ESDM should be optional for small projects.
- Personnel qualifications for expert systems development should include experience and familiarity with graphics user interfaces as well as with the functional tools required for expert systems.

REFERENCES

Boehm, B. (May 1988). A spiral model of software development. *IEEE Computer.*

Bush, J. L. and Weaver, S. J. (May 1989). BCAUS project description and consideration of separation of data and control. *NASA Conference Publication 3033.*

Computer Sciences Corporation (a) (September 1988). Expert system development methodology (user's guide), DSTL-90-004.

-- (b) (September 1988). Expert system development methodology (policy document), DSTL-90-005.

-- (c) (September 1988). Expert system development methodology (reference manual), DSTL-90-006.

-- (March 1989). Expert system development methodology; Framework for evaluation of ESDM.

Luczak, E. C., Gopalakrishnan, K., and Zillig, D. (May 1989). REDEX: The ranging equipment diagnostic expert system. *NASA Conference Publication 3033.*

SATELLITE IMAGE ANALYSIS USING NEURAL NETWORKS

Roger A. Sheldon
Ford Aerospace
Seabrook, MD

ABSTRACT

The tremendous backlog of unanalyzed satellite data necessitates the development of improved methods for data cataloging and analysis. Ford Aerospace has developed an image analysis system, SIANN, that integrates the technologies necessary to satisfy NASA's science data analysis requirements for the next generation of satellites. SIANN will enable scientists to train a neural network to recognize image data containing scenes of interest and then rapidly search data archives for all such images. The approach combines conventional image processing technology with recent advances in neural networks to provide improved classification capabilities. SIANN allows users to proceed through a four step process of image classification: filtering and enhancement, creation of neural network training data via application of feature extraction algorithms, configuring and training a neural network model, and classification of images by application of the trained neural network. A prototype experimentation testbed has been completed and applied to climatological data.

INTRODUCTION

Data acquired from satellites are essential resources in meteorology, agriculture, astronomy, forestry, geology, oceanography, and many other fields. Cataloging and analysis of image data has been a fundamental problem for NASA. For instance, in 1986 a team of scientists at the South Pole took readings overhead and learned that the "hole" in the Earth's ozone was getting worse. It was later discovered that the hole actually showed up in 1976 in Nimbus 7 satellite data. Concerning this discovery, James L. Green, head of the NASA National Space Science Data Center stated in (Kneale, 1988), "It's one of probably hundreds of important discoveries we have sitting in the basement." To compound this problem, the next generation of scientific satellites will generate far greater amounts of data.

How will such an enormous database be accessed, and how will large amounts of data be analyzed? To help provide solutions to these questions, Ford Aerospace is investigating neural network technology to determine how it can provide improved satellite image analysis capabilities. A prototype system called SIANN (Satellite Image Analysis using Neural Networks) has been developed which combines conventional image processing techniques with neural networks. Currently, SIANN addresses the image cataloging problem; that is, the generation of summary information, or "metadata", from raw image data. The metadata are stored in a database which will enable scientists to rapidly retrieve images containing scenes of interest.

SIANN is intended to be a general-purpose classification system. It will be embedded into large satellite data management systems and provide a library of feature extraction and classification programs to support dozens of scientific disciplines.

Scientists working in different domains may be interested in the same data. As such, it is necessary to apply a variety of algorithms to the raw image data to create the metadata that will support queries from multiple scientific domains.

Scientists develop classifiers in SIANN by using the following procedure, which is illustrated in figure 1:

1) Select (or develop if necessary) feature extraction algorithms which are appropriate for the scientific

discipline of interest and create a training set, T, of patterns representative of the desired classes

2) Configure and train a supervised learning neural network to identify the desired classes of image scenery using the training set T

3) Test the classifier by applying it to novel data; if the results are not satisfactory, repeat steps 1 and/or 2 and then retest.

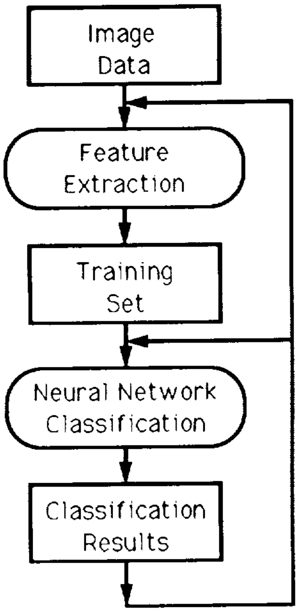


Figure 1. The SIANN image classification process. Rectangles represent data and ovals represent operations.

The creation of neural network training sets by applying feature extraction algorithms has proven successful in a number of different applications (Rimey, 1986; Beck, 1989). Another useful approach is to classify individual pixels from multispectral images (Campbell, 1989; McClellan, 1989).

This paper presents initial results of SIANN applied to climatological image data. First, the feature extraction process is described. Next, the neural network classification technique is presented. Then an experiment is described which analyzes the effects that varying the number of training set features has on a neural network's classification correctness. Finally, conclusions are made and directions of future work are stated.

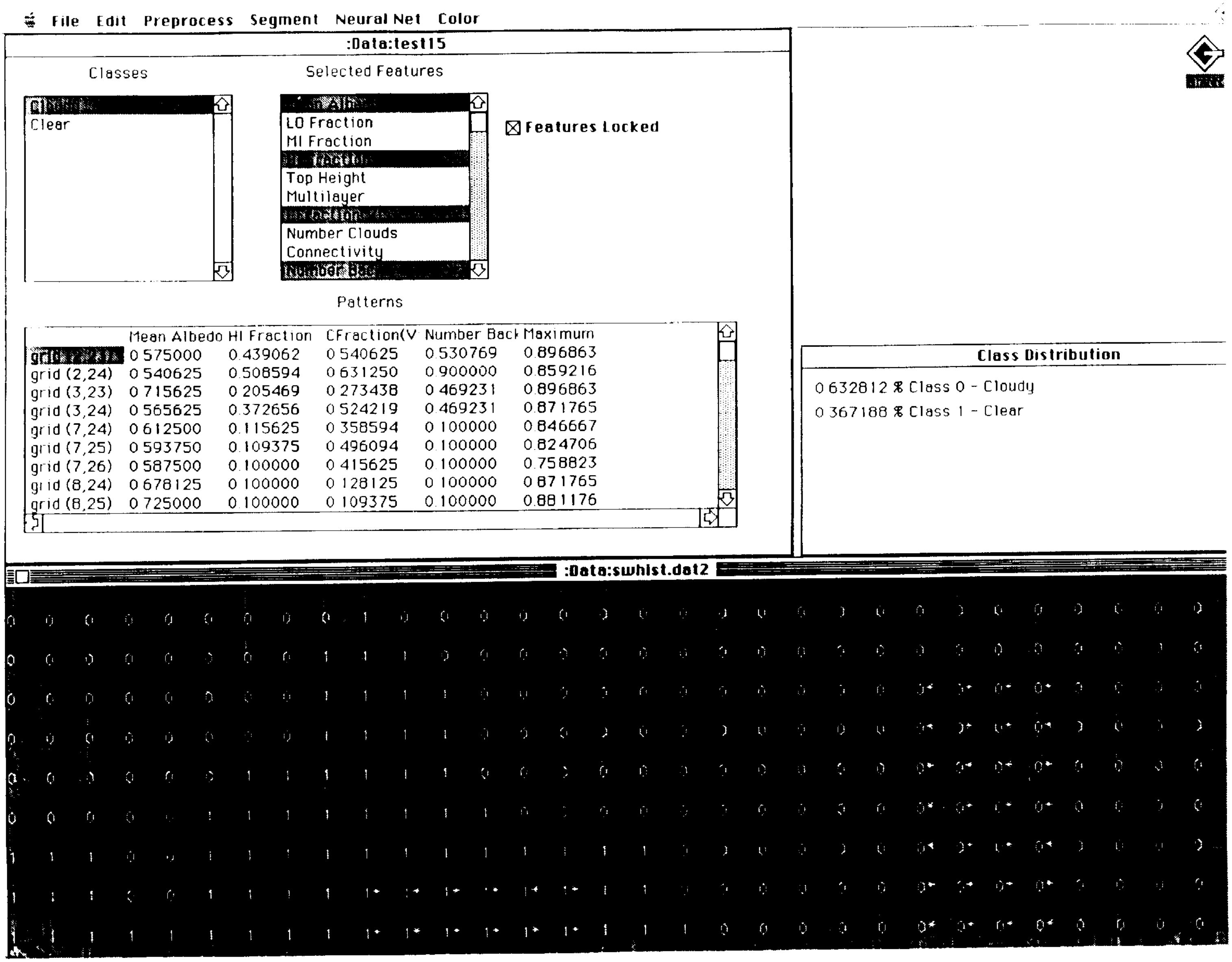
FEATURE EXTRACTION

(Garand, 88) describes 13 features representing height, albedo, shape, and multilayering characteristics of cloud fields. Table 1 lists 12 of these features (Garand's feature for 'Fraction of cloudy pixels with D $< D_r$' was not included.) plus three simple statistical features and the 'Number background' feature which is a variation of the 'Number of clouds' feature.

Table 1. Features used to classify climatological data. Image source: VIS, visible; IR, infrared; B&W, binary corresponding to visible cloud fraction; PS, power spectrum.

	Description	Limits
1.	Total cloud fraction (IR, VIS)	0-1
2.	Low cloud fraction (IR)	0-1
3.	Middle cloud fraction (IR)	0-1
4.	High cloud fraction (IR)	0-1
5.	Cloud height of uppermost layer (IR)	0-14 km
6.	Fraction of cloudy pixels (VIS)	0-1
7.	Mean albedo of cloudy pixels (VIS)	0-1
8.	Number of clouds (B&W)	0-*m*/2
9.	Multilayer index (IR)	0-1
10.	Background connectivity (B&W)	0-1
11.	Cloud connectivity (B&W)	0-1
12.	Streakiness factor (PS)	0-1
13.	Fraction of spectral intensity associated with wavelengths between 20-40 km (PS)	0-1
14.	Minimum pixel value (VIS)	0-255
15.	Maximum pixel value (VIS)	0-255
16.	Range of pixel values (VIS)	0-255
17.	Number background (VIS)	0-*m*/2

Note that Garand's work is solely directed at the classification of 20 cloud types, without regard to computation time. Since classifiers created from SIANN will be applied to immense databases, there is usually a time/accuracy tradeoff. That is, computationally inexpensive features are



☒ Features Locked

	Mean Albedo	HI Fraction	CFraction(V	Number Bacl	Maximum
grid	0 575000	0.439062	0 540625	0 530769	0.896863
grid (2,24)	0 540625	0.508594	0 631250	0 900000	0.859216
grid (3,23)	0 715625	0 205469	0 273438	0 469231	0.896863
grid (3,24)	0 565625	0.372656	0 524219	0 469231	0.871765
grid (7,24)	0 612500	0.115625	0 358594	0 100000	0.846667
grid (7,25)	0 593750	0.109375	0 496094	0 100000	0.824706
grid (7,26)	0 587500	0.100000	0 415625	0 100000	0.758823
grid (8,24)	0 678125	0 100000	0 128125	0 100000	0.871765
grid (8,25)	0 725000	0 100000	0 109375	0 100000	0.881176

Class Distribution

0 632812 % Class 0 - Cloudy

0 367188 % Class 1 - Clear

Figure 2. SIANN enables scientists to create a neural network training set by specifying the desired features and classes, and then selecting image regions representative of the classes.

usually selected over more accurate, but more expensive features. Likewise, smaller neural networks (i.e., fewer nodes) are preferred to larger networks.

Figure 2 shows the SIANN user interface for creating a training set. The first step is to enter the desired classes of image scenes into the "Classes" box. Next, a set of features are selected from the "Features" box. Then image regions that are representative of the classes are selected as depicted by the starred ("*") regions in figure 2. Alternatively, odd sized regions may be selected in addition to the fixed-sized grid regions. Finally, a command is issued to generate the feature vectors for the selected image regions. The "Patterns" box shown in figure 2 contains the feature vectors generated from the selected image regions (but only for the currently selected class in the "Classes" box). Each feature vector consist of several real numbers typically in the range of 0...1, which are used as inputs to the neural network. Each class defined by the user is represented by one output of the neural network. For the remainder of this paper, the training set shown in figure 2 shall be referred to as the "test" training set.

NEURAL NETWORK CLASSIFICATION

SIANN uses the popular backpropagation algorithm (Rumelhart, 1986). Figure 3 illustrates the general topology of a back propagation network. The bottom layer of nodes is the input layer where patterns are presented to the network. The top layer contains the output nodes which indicate the class of the input pattern. Any number of internal layers are permitted, but typically one is sufficient. (The paper by Ho, 1989 concludes that it is generally better to increase the width of the network than to add layers.)

Given a training set, SIANN will automatically configure and initialize a network. Figure 4 describes the network that SIANN generated from the test training set. (This network will be referred to as the "test" network.) Note that the number of input nodes matches the number of features and the number of output nodes matches the number of classes. Each feature value of the input patterns is scaled from the corresponding limits in table 1 to the range 0.1 to 0.9.

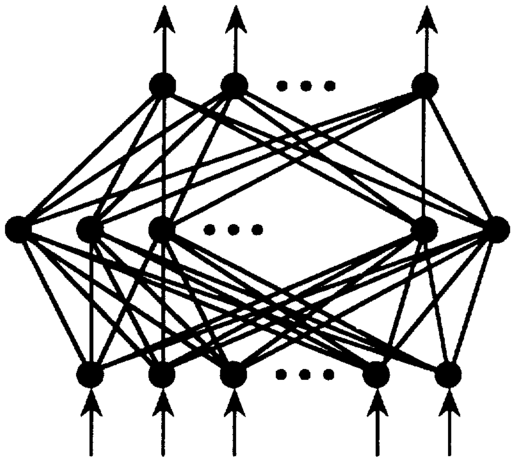
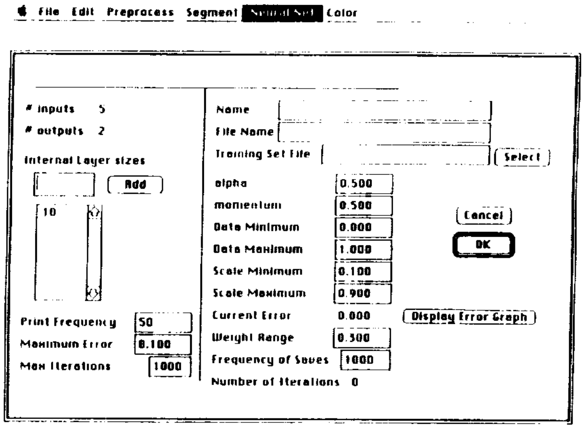


Figure 3. A back propagation neural network receives a pattern in its bottom, input layer and computes the pattern's class in the top, output layer.



inputs 5
outputs 2
Internal Layer sizes
10
Print Frequency 50
Maximum Error 0.100
Max Iterations 1000

Name
File Name
Training Set File
alpha 0.500
momentum 0.500
Data Minimum 0.000
Data Maximum 1.000
Scale Minimum 0.100
Scale Maximum 0.900
Current Error 0.000
Weight Range 0.300
Frequency of Saves 1000
Number of Iterations 0

Figure 4. SIANN automatically configures a back propagation neural network to train on a specified training set.

Before training begins, SIANN automatically creates unary vectors for the target outputs. For the 2-class test network, the vectors are:

Class 0:	0.1	0.9
Class 1:	0.9	0.1

After training, the network is tested by applying it to new data (i.e., data that it wasn't trained with). Figure 5 shows the results of classifying the test image with the test network. Misclassifications are denoted by an "X*i*" where X means WRONG, and *i* is the class computed by the network. If the network does not perform satisfactorily, the scientist may modify the training set and/or network and then retest.

EXPERIMENTAL RESULTS

The data set used contains three 1024 x 384 8 bit AVHRR (Advanced Very High Resolution Radiometer) images of the Indian Ocean. Each 8 bit pixel has a footprint of 1 km^2. For purposes of this discussion, we shall focus on the image shown in figure 2, which has been overlaid with the author's subjective classification of the picture. (This image will be referred to as the "test" image.)

Table 2. Six training sets were created containing from 4 to 9 features.

	Training Set					
Feature	1	2	3	4	5	6
Mean albedo	•	•	•	•	•	•
Low cloud fraction	•	•	•	•	•	•
Middle cloud fraction	•	•	•	•	•	•
High cloud fraction	•	•	•	•	•	•
Cloud top height		•	•	•	•	•
Cloud fraction			•	•	•	•
Number background				•	•	•
Maximum pixel value					•	•
Range of pixel values						•

An experiment was conducted to determine the effects of modifying the set of selected features. Six 2-class training sets were created, each containing patterns from the size 32^2 grid regions lying in the rectangle whose upper left grid coordinates are (4, 10) and lower right grid coordinates are (7, 20). This provided 25 Cloudy patterns and 19 Clear patterns for each training set. The features used in each training set are listed in table 2. Each training set was used to train a network. Each network had a single internal layer of 15 nodes. Equation (1) was used during training to modify each weight, where the learning rate $\alpha = 0.9$, and the momentum term $\eta = 0.7$.

$$w_{ij}(t+1) = w_{ij}(t) + \eta\delta_j x_j + \alpha(w_{ij}(t)-w_{ij}(t-1)) \tag{1}$$

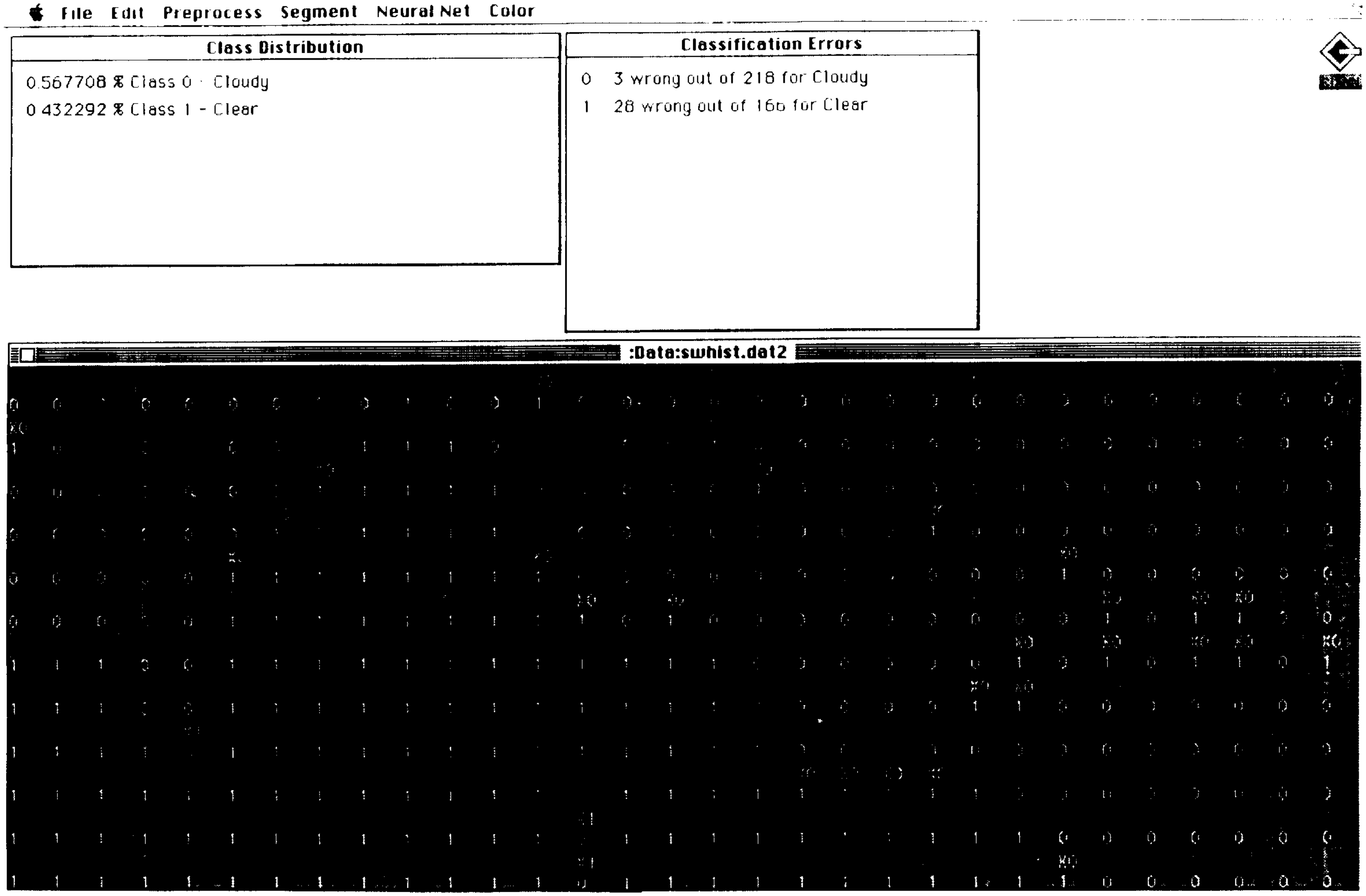
Convergence for each network occurred when the maximum error fell below 0.1. Table 3 summarizes the results of training the networks. The third column specifies how many training iterations each network required to converge. The fourth column lists the CPU time of each training run on a VAXstation 3540. Note that when the number of features decrease, the time for each iteration also decreases since the number of nodes in the network is reduced.

Table 3. Training times increase as the number of input features decreases.

Training Set	Number Features	Number Iterations	Minutes
1	4	1790	5.6
2	5	2278	7.8
3	6	711	2.7
4	7	351	1.5
5	8	316	1.3
6	9	456	2.1

The nonlinearity of the number of iterations and minutes for training runs can be attributed to the characteristics of certain features. Specifically, it would appear that the addition of the 'Cloud top height' feature that distinguishes training set 1 from training set 2 detracts from the separability of patterns within training set 2. Similarly, the addition of the 'Range of pixel values' feature detracts from the separability of training set 6.

Each network was tested by applying it to all 384 grid regions of the test image. Figure 6 plots the percentage of misclassified regions vs. the number of features. The number of classification errors tends to decrease when a larger number of input features are used.



Class Distribution

0.567708 % Class 0 · Cloudy
0 432292 % Class 1 - Clear

Classification Errors

0 3 wrong out of 218 for Cloudy
1 28 wrong out of 166 for Clear

:Data:swhist.dat2

Figure 5. The neural network's classifications of the image regions are compared to the scientists classifications to help refine the classification process.

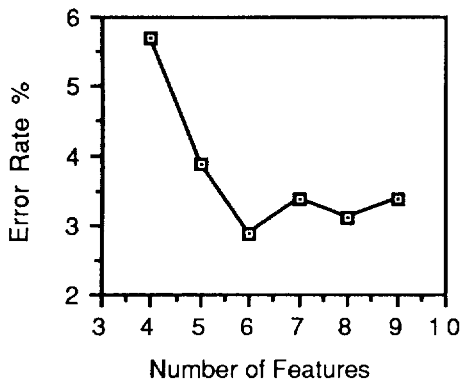


Figure 6. Neural network classification errors tend to increase as the number of input features decreases.

Scientists using SIANN can analyze training and classification results to select the optimal set of features and most efficient neural network configuration.

CONCLUSIONS

SIANN is an image analysis system that combines conventional image processing techniques with neural network classifiers. Scientists may quickly develop a customized classifier using SIANN's menu-driven graphical interface. Analysis of the classifier's behavior helps the scientist improve the classifier by modifying the training set features and neural network configuration.

Preliminary testing of the system on climatological data has demonstrated that neural networks are a viable technique for image analysis. SIANN will continue to evolve by adding feature extraction programs for other scientific domains. Another future direction is to investigate unsupervised learning neural networks to determine if the classifier refinement process can be automated; that is, to see if a network can discover by itself a good set of features.

REFERENCES

Beck, H., McDonald, D., & Brzakovic, D. (1989). A self-training visual inspection system with a neural network classifier. *Proceedings from the 1989 International Joint Conference on Neural Networks* (pp. I-307 - I-311). Washington, D.C.

Campbell, W., Hill, S., & Cromp, R. (1989). The utilization of neural nets in populating an object-oriented database. *Proceedings from the 1989 Goddard Conference on Space Applications of Artificial Intelligence.* (pp. 249-263). Greenbelt, MD.

Garand, L. (1988). Automated recognition of oceanic cloud patterns. Part I: Methodology and application to cloud climatology. *Journal of Climate*, 1(1), 20-39.

Ho, C. (1989). On multi-layered connectionist models: Adding layers vs. increasing width. *Proceedings from the Eleventh International Joint Conference on Artificial Intelligence.* (pp. 176-179). Washington, D.C.

Kneale, D. (January 11, 1988). What becomes of data sent back from space? Not a lot, as a rule. *Wall Street Journal,* p. 1.

McClellan, G., DeWitt, R., Hemmer, T., Matheson, L., & Moe, G. (1989). Multispectral image-processing with a three-layer backpropagation network. *Proceedings from the 1989 International Joint Conference on Neural Networks* (pp. I-151 - I-153). Washington, D.C.

Rimey, R., Gouin, P., Scofield, C., & Reilly, D. (1986). Real-time 3-D object classification using a learning system. *SPIE Intelligent Robots: 6th International Conference on Robot Vision and Sensory Controls, RoViSeC, SPIE Vol. 726.* (pp. 552-557). Cambridge, MA.

Rumelhart, D., & McClelland, J. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* Cambridge, MA: MIT Press.

National Aeronautics and Space Administration

Report Documentation Page

1. Report No. NASA CP-3068	2. Government Accession No.	3. Recipient's Catalog No.

4. Title and Subtitle

1990 Goddard Conference on Space Applications of Artificial Intelligence

5. Report Date

May 1990

6. Performing Organization Code

Code 500

7. Author(s)

James L. Rash, Editor

8. Performing Organization Report No.

90B00078

10. Work Unit No.

9. Performing Organization Name and Address

Mission Operations & Data Systems Directorate
NASA/GSFC, Code 500
Greenbelt, MD 20771

11. Contract or Grant No.

12. Sponsoring Agency Name and Address

National Aeronautics & Space Administration
Washington, DC 20546

13. Type of Report and Period Covered

Conference Publication

14. Sponsoring Agency Code

Code 500

15. Supplementary Notes

James L. Rash is associated with Goddard Space Flight Center, Greenbelt, Maryland.

16. Abstract

This publication comprises the papers presented at the 1990 Goddard Conference on Space Applications of Artificial Intelligence held at the NASA/Goddard Space Flight Center, Greenbelt, Maryland on May 1-2, 1990. The purpose of this annual conference is to provide a forum in which current research and development directed at space applications of artificial intelligence can be presented and discussed. The papers in this proceedings fall into the following areas: Planning & Scheduling, Fault Monitoring/Diagnosis, Image Processing & Machine Vision, Robotics/Intelligent Control, Development Methodologies, Information Management, and Knowledge Acquisition.

17. Key Words (Suggested by Author(s))

Artificial Intelligence, expert systems, planning and scheduling, fault isolation, fault diagnosis, image processing, machine vision, data management, robotics, control, knowledge acquisition.

18. Distribution Statement

Unclassified - unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	368	A16

NASA-Langley, 1990

National Aeronautics and
Space Administration
Code NTT-4

Washington, D.C.
20546-0001

Official Business
Penalty for Private Use, $300

NASA

POSTMASTER: [illegible]