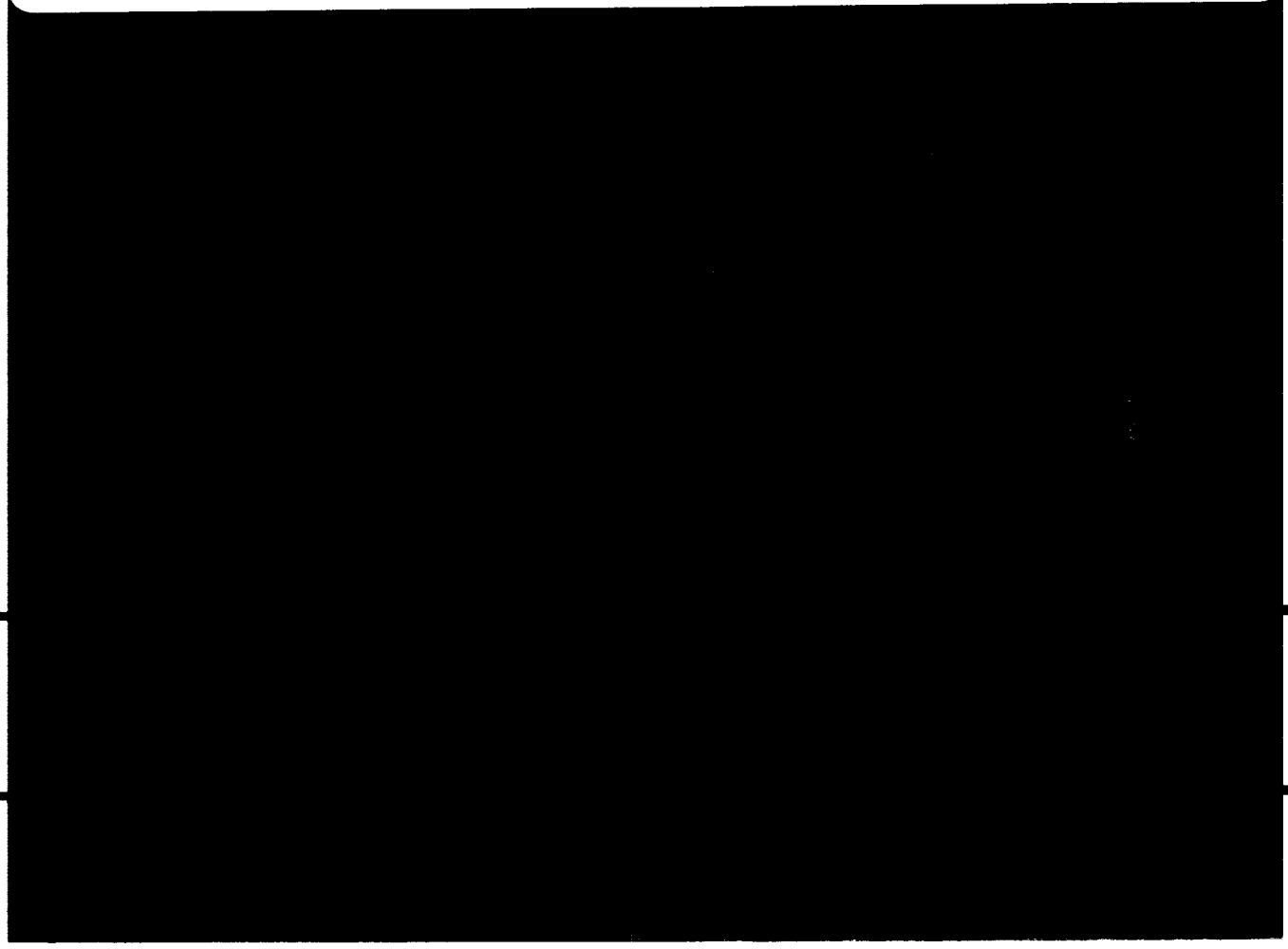
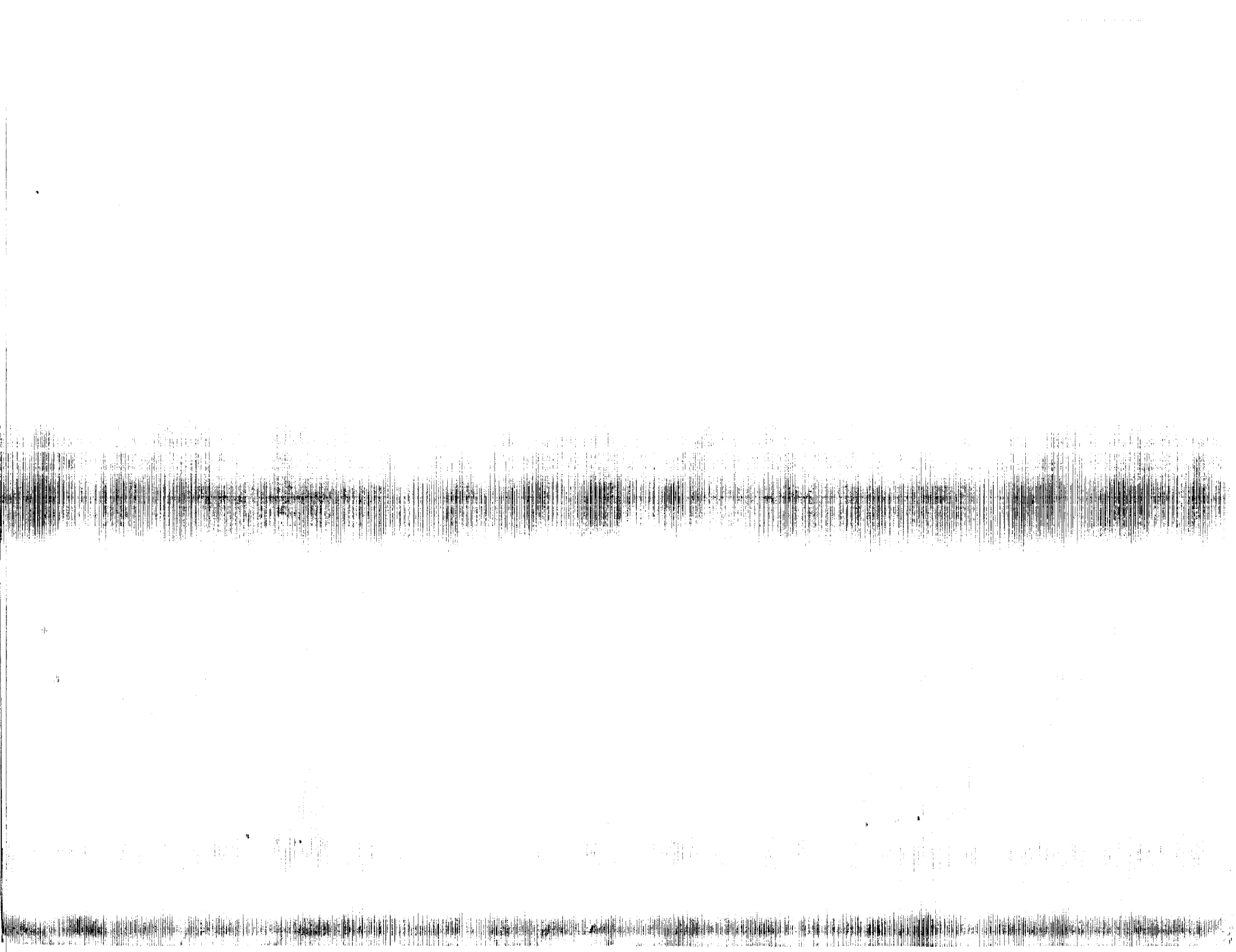
NASA Conference Publication 3141

1992 Goddard Conference on Space Applications of Artificial Intelligence





NASA Conference Publication 3141

1992 Goddard Conference on Space Applications of Artificial Intelligence

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

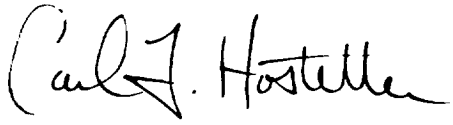
Proceedings of a conference held at
Goddard Space Flight Center
Greenbelt, Maryland
May 5–6, 1992

NASA
National Aeronautics and
Space Administration
Office of Management
Scientific and Technical
Information Program
1992

Foreword

The seventh 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 Conference provides a needed and effective forum for the exchange of ideas, techniques, and experiences among the researchers and practitioners of Artificial Intelligence throughout the space industry.

No conference simply "happens", even one that has had six previous iterations of prototyping and testing. Thus, as always, there is the pleasant duty of acknowledging the many people that have labored to bring about this year's Conference. Thanks, of course, to the authors, whose work makes our Conference possible and worthwhile; and to the invited speakers and panelists, for sharing their time and insight with us. Thanks, too, to our Paper Review Panel, new to this year's Conference, for their time and expertise. And, as always, special thanks to the Conference Planning Committee for their dedication and for the hard, sometimes frantic work that lies behind every aspect of the Conference, and that the Conference depends upon for its existence.



Carl F. Hostetter
Chairman, 1992 Goddard Conference
on Space Applications of Artificial Intelligence

Preface

This proceedings of the seventh annual Goddard Conference on Space Applications of Artificial Intelligence, held May 5 and 6, 1992 at the Goddard Space Flight Center, Greenbelt, Maryland, offers nineteen papers demonstrating progress in the areas of planning and scheduling, monitoring/diagnosis/control, tools, information management, neural networks, and certain miscellaneous applications.

By virtue of an expanded paper selection process and a new editing cycle, we believe this year's proceedings reaches a high point in quality. All of the submitted papers were selected through blind review by a selection committee drawn from NASA, industry, and academia. During the editing cycle, time limitations permitted only minor revisions. The entire process, considered to be worth the necessary additional time and effort, is likely to be repeated, or extended even further, for next year's conference.

Under sponsorship of Goddard's Mission Operations and Data Systems Directorate, the conference remains one of the most accessible of all the means available to AI practitioners for communicating results relative to space applications of artificial intelligence. No admission fees are charged, and attendance is open to all U.S. citizens and qualified noncitizens. We continue broadly soliciting contributions of technical papers appropriate to the theme of the conference. Potential contributors should take note of the call for papers for the 1993 conference printed at the end of this proceedings document.

James L. Rash
Document Editor

Table of Contents

Image/Data Classification/Interpretation

Knowledge Engineering

Information Management

Planning and Scheduling

N92-23357

Artificial Intelligence Approach to Planning the Robotic Assembly of Large Tetrahedral Truss Structures*

Luiz S. Homem-de-Mello
Jet Propulsion Laboratory
California Institute of Technology
Pasadena CA 91109-8099

Abstract

This paper presents an assembly sequence planner for tetrahedral truss structures. To overcome the difficulties due to the large number of parts, the planner exploits the simplicity and uniformity of the shapes of the parts and the regularity of their interconnection. The planning automation is based on the computational formalism known as *production system*. The global database consists of an hexagonal grid representation of the truss structure. This representation captures the regularity of tetrahedral truss structures and their multiple hierarchies. It maps into quadratic grids and can be implemented in a computer by using a two dimensional array data structure. By maintaining the multiple hierarchies explicitly in the model, the choice of a particular hierarchy is only made when needed, thus allowing a more informed decision. Furthermore, testing the preconditions of the production rules is simple because the patterned way in which the struts are interconnected is incorporated into the topology of the hexagonal grid. A directed graph representation of assembly sequences allows the use of both graph search and backtracking control strategies.

1 Introduction

Figure 1 shows a tetrahedral truss structure similar to those that will be used in future space missions [13]. The assembly, disassembly or repair of these large truss structures requires careful planning in order to guarantee that the parts are assembled, or disassembled, in a correct and efficient sequence. This planning is needed regardless of whether the assembly is executed by humans or by robots.

Because of the size and complexity of these truss structures, even trained humans may fail to detect *dead-end* sequences until a lot of work has been done and it is found that the overall assembly cannot be completed. In the case of a repair in which a faulty strut is to be replaced, an ill-planned disassembly sequence may lead to an irreparable collapse of the whole truss structure.

In addition to the difficulty humans have in guaranteeing correctness in the planning process, they often fail to notice which possibilities for the sequences are the most efficient. This difficulty is further aggravated by constant changes in the measure of the efficiency of the assembly sequence. For example, the efficiency may be measured by the total time it takes to complete the assembly in one case, and by the total energy in another case.

Moreover, humans typically are slow in generating assembly sequences. There are many situations in which the sequence planning must also be expeditious. Speed in sequence generation is particularly important in the case of a repair in which a faulty strut is to be replaced. It is virtually impossible to preplan for every conceivable repair that may be needed. Speed in sequence generation is also important in the design of the truss structures because it allows the difficulty of assembly to be considered in the design process. A designer may also want to take into account the difficulty of repairing different structures.

Therefore, there is a need to systematize and to computerize the generation of the correct assembly, disassembly, and repair sequences, as well as the selection of the best solution. Systematization is needed in order to guarantee that the sequences generated are correct and efficient. Computerization is needed in order to enable the fast generation of assembly sequences. In the case of robotic assembly and repair, the software for planning assembly, disassembly, and repair sequences will augment the array of functions that robots are able to perform autonomously.

Most previous work on assembly planning focused on electromechanical and electronic devices [6]. This paper presents an assembly sequence planner for tetrahedral truss structures. To overcome the difficulties due to the large number of parts, the planner exploits the simplicity and uniformity of the shapes of the parts and the regularity of their interconnection. The planning automation is based on the computational formalism known as *production system*. The

*This research was conducted at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

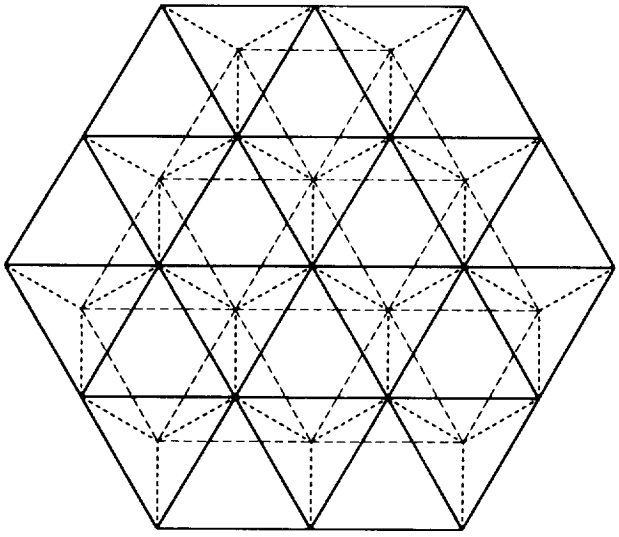


Figure 1: Tetrahedral truss structure.

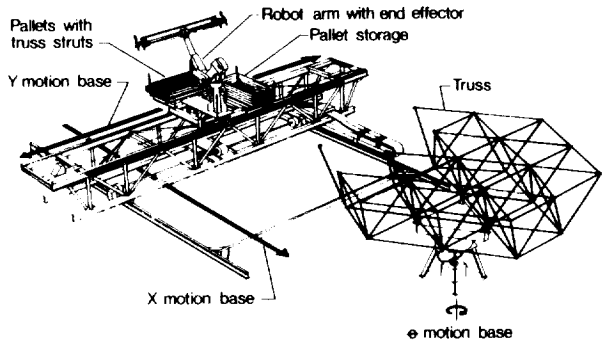


Figure 2: Robotic assembly facility at the NASA Langley Research Center.

global database consists of an hexagonal grid representation of the truss structure. This representation captures the regularity of tetrahedral truss structures and their multiple hierarchies. It maps into quadratic grids and can be implemented in a computer by using a two dimensional array data structure. By maintaining the multiple hierarchies explicitly in the model, the choice of a particular hierarchy is only made when needed, thus allowing a more informed decision. Furthermore, testing the preconditions of the production rules is simple because the patterned way in which the struts are interconnected is incorporated into the topology of the hexagonal grid. A directed graph representation of assembly sequences allows the use of both graph search and backtracking control strategies.

2 Scenario

A truss structure is a composition of struts interconnected at nodes and forming a stable and rigid unit. All struts are identical, and so are all nodes. The struts are attached to the nodes through joint connectors. A strut s_i is said to belong to a node n_j if one of s_i's ends is attached to n_j. Similarly, node n_j is said to belong to strut s_i.

In this paper, the robotic assembly facility of the NASA Langley Research Center [12, 13, 15] will be used as the reference scenario. Figure 2 shows that facility in schematic form. The robot arm is mounted on a base that is mounted on a carriage that can translate along one direction. The base where the robot is mounted can translate along a direction orthogonal to the carriage translations. These two motions allow the positioning of the robot arm in a Cartesian coordinate system. The truss structure is mounted on a base that can rotate. If necessary, before a strut is assembled, the structure is turned and both the base of the robot arm and the carriage are translated.

The assembly process consists of a succession of tasks, each of which is the addition of one strut. The nodes are preattached to their first strut to be assembled. Whenever a strut is added, it is attached to its two nodes, except when the nodes have been preattached. The process starts with all struts stored in pallets that are stacked on the same base where the robot arm is mounted. The assembly process ends with all struts properly joined to form the whole structure. Ideally, after struts have been added, they are not removed until the end of the assembly process.

An assembly task is said to be feasible if there is a collision-free path to bring the strut to its position in the structure from a situation in which it is far apart, and if it is possible to lock the joints that attach the strut to its nodes. Of course, the path should also avoid collisions between the robot arm or the carriage and the truss structure.

It is desired to create a computer system that will generate a sequence of assembly tasks for any given tetrahedral truss structure. Of course, the input to this system includes a description of the desired structure. In addition to containing only feasible tasks and achieving the assembly of the whole structure, the sequence produced should minimize a given cost function. Although the definition of the cost functions is part of the problem, it will probably include a weighed combination of reliability, safety, energy and total assembly time.

3 Background on assembly sequence planning

Most previous work on assembly sequence planning [2, 3, 4, 5] focused on electromechanical and electronic devices such as gearboxes, alternators and disk drives. The difficulty in planning the assembly sequence for those products stems, in some degree, from the variety of part shapes and from the lack of regularity in the way the pieces are interconnected. To overcome this difficulty, previous approaches used elaborate representations of mechanical assemblies and complex geometric and symbolic reasoning techniques [1, 4, 16].

Another difficulty in the automation of assembly sequence planning for electromechanical and electronic devices comes from the fast growth in the required computation with the increase in the number of parts. Previous approaches have overcome this problem by clustering components into subassemblies [8], thereby artificially reducing the number of parts. Many large products have natural subassemblies that arise as a result of modular design as well as of manufacturing advantages. Clustering components into subassemblies sacrifices completeness since sequences that interleave the assembly of parts of different subassemblies can not be generated. But for most large products this loss of completeness is not a serious limitation because those natural subassemblies are assembled independently anyway. In practice, a hierarchical model of the assembly [14] is used to implement the clustering of parts. At the highest level, each subassembly is treated as a part.

There has been substantial progress in assembly planning in recent years, and several new approaches and techniques have been reported[6]. Nevertheless, because of the complexity of the reasoning involved and the large size of the solution space, it is still impracticable to use existing planners to generate an assembly sequence for assemblies containing a large number of parts, such as the structure shown in Figure 1, which is made of 102 struts.

One way to reduce the computation, when planning the assembly of tetrahedral truss structures, is to cluster the struts into subassemblies as in the case of electromechanical and electronic devices. A truss structure such as the one shown in Figure 1 can be viewed as the composition of tetrahedral and pentahedral units, much like a solid that has a complex shape can be treated as the composition of simple solids that have faces in contact, one against the other. Two adjacent units share the struts and nodes of their "contacting" faces. For example, the small truss structure shown in Figure 3a can be regarded as the composition of the two pentahedral units shown in Figure 3b. Those two pentahedrons have one face "in contact" and they share the struts and nodes of that face. Similarly, the structure shown in Figure 1 can be viewed as a composition of tetrahedral and pentahedral units with faces "in contact."

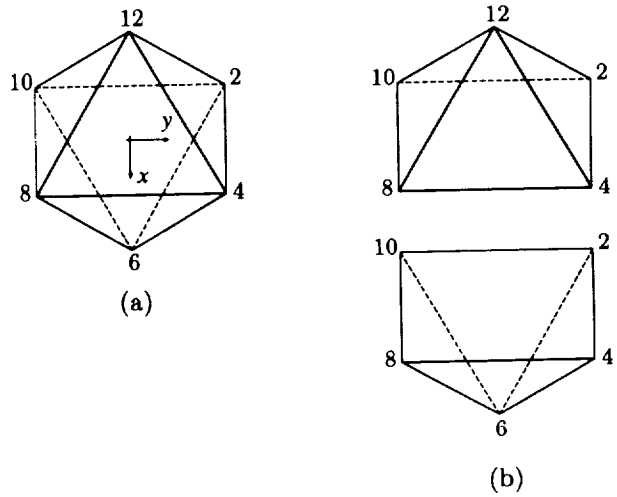


Figure 3: A small truss structure and its subdivision into two pentahedral units.

Since in practice it is preferred to finish the assembly of one unit before beginning the assembly of another [11], seeing the structure as a composition of units should not be a problem in the assembly sequence planning. Furthermore, this approach should reduce the computation required to create the assembly sequence. Mathur and Sanderson [9] describe a hierarchical planner for truss structures. For the structure shown in Figure 1, for example, using this approach corresponds to reducing the number of parts from 102, which is the number of struts, to 37, which is the number of units.

The use of a hierarchical approach for planning the assembly requires that the tetrahedral truss structure be subdivided into pentahedral and tetrahedral units. But there are several ways to cluster the struts into those kinds of unit. For example, in addition to the subdivision shown in Figure 3b, there are two other ways in which the small truss structure shown in Figure 3a can be subdivided into two pentahedral units, and these are shown in Figure 4. Unlike the electromechanical and electronic devices studied previously, in the case of truss structures, there is no manufacturing advantage in choosing one subdivision over the others. Instead of having one natural hierarchy of the parts, tetrahedral truss structures have several hierarchical models, none of which is "more natural" than the others. When the small structure shown in Figure 3 is part of a large structure (e.g. Figure 1), unless the best assembly sequence is known in advance, choosing one of its subdivisions to create a hierarchical model will likely preclude the generation of the best assembly sequence. Therefore, it is important that the representation of the problem captures the multiple hierarchies that occur in a tetrahedral

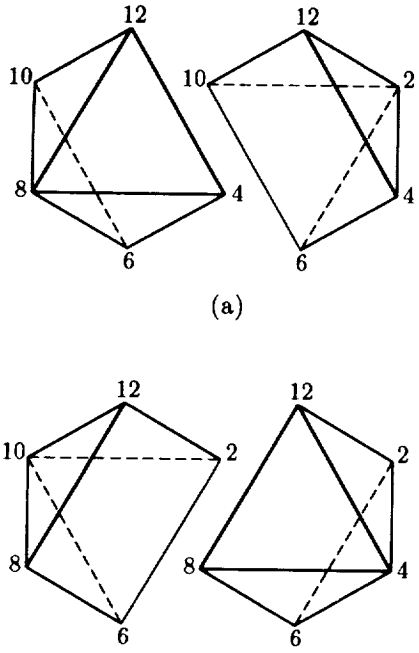


Figure 4: Two additional subdivisions of the small truss structure (Figure 3a) into two pentahedral units.

truss structure.

A second way to reduce the computation, when planning the assembly of tetrahedral truss structures, is to take advantage of the simplicity and uniformity of the shapes of the parts and the regularity of their interconnection. Unlike the electromechanical and electronic devices studied previously, the tetrahedral truss structures are made of struts, all of which have the same cylindrical shape. Moreover, those struts are interconnected in a regular fashion. Because the parts have the same shape and are interconnected in a patterned way, the model of a truss structure can incorporate the geometry of the set of parts in a more explicit way than the models used for electromechanical and electronic devices.

The models of assemblies that have been used in previous work describe the shapes of all parts and the geometric and mechanical relationships between parts. Typically, assembly models can be associated with graphs in which the vertices correspond to the parts and the edges to the geometric relationships between parts [14]. The topology of the graph corresponds to the topology of the parts in the assembly. But there is no relation between the geometry of the set of parts in the assembly and the topology of the graph. Figure 5 shows three simple assemblies made up of the same set of parts. Those assemblies are associated with the same graph, also shown in Figure 5, since the topology of the parts is the same. But the

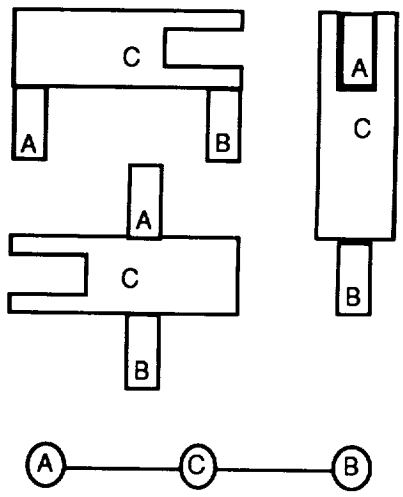


Figure 5: Three assemblies with different geometry but same topology.

geometry of the parts in each assembly is very different from the geometry of the parts in the others.

The next section presents an assembly planner that uses a multihierarchical representation for the truss structures and that takes advantage of the simplicity and uniformity of the shapes of the parts and the regularity of their interconnection.

4 Planning the assembly of tetrahedral truss structures

The computational formalism known as production system [10] has been used for the automatic generation of assembly sequences for tetrahedral truss structures. There are three major elements in a production system: the global database, the set of production rules, and the control scheme. This section describes these three elements. Subsection 4.1 presents a multihierarchical representation of tetrahedral truss structures that constitutes the global database; subsection 4.2 discusses the control scheme; and subsection 4.3 introduces the production rules that act on the global database.

4.1 A multihierarchical representation of tetrahedral truss structures

It was mentioned in section 3 that, unlike the electromechanical and electronic devices studied previously, the tetrahedral truss structures are made of simple struts all of which have the same shape. In addition, the struts are interconnected in a very regular

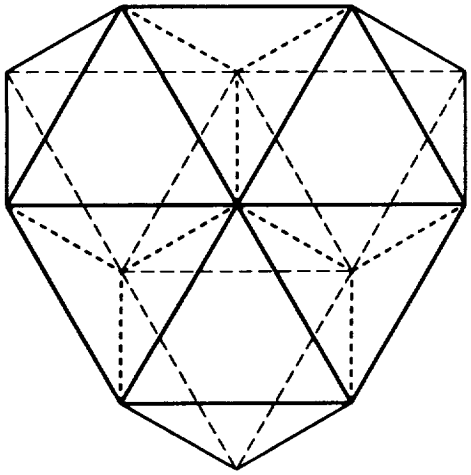


Figure 6:A subset of the structure shown in Figure 1.

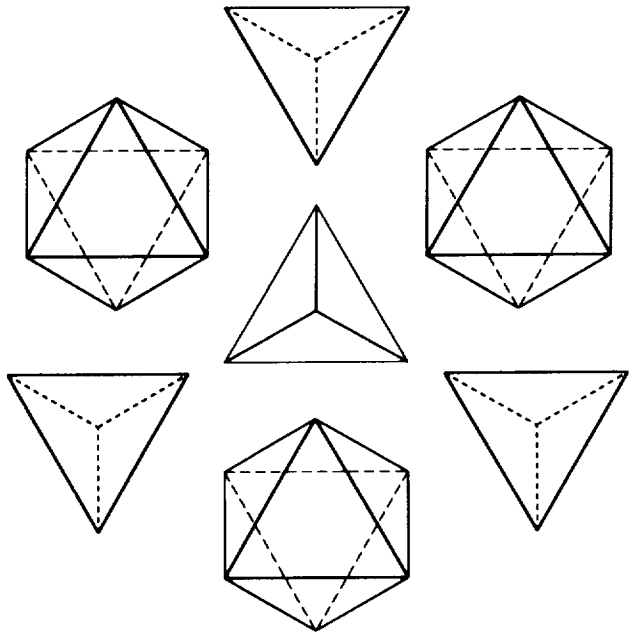


Figure 7: The building blocks of the structure shown in Figure 6.

fashion. The method for planning assembly sequences of tetrahedral truss structures can take advantage of these facts to reduce the computation needed to generate assembly sequences.

The representation to be introduced next is based on viewing tetrahedrons and octahedrons as the building blocks of a tetrahedral truss structure. A pentahedron will be considered to be a building block only when it is not embedded in any octahedron[1]. Figure 6 shows a subset of the structure depicted in Figure 1, and Figure 7 shows its building blocks.

As discussed above, there are six embedded pentahedrons in an octahedron. The nodes of the octahedron shown in Figure 3a have been numbered by analogy with the numbers in a clock face. The embedded pentahedrons are designated by the number of the vertex corresponding to their apex. Therefore, the pentahedrons shown in Figure 3b are referred to as P12 (top) and P6 (bottom); the pentahedrons shown in Figure 4a are referred to as P8 (left) and P2 (right); and the pentahedrons shown in Figure 4b are referred to as P10 (left) and P4 (right).

Figure 3a shows a coordinate frame associated with an octahedron. The x–y plane contains nodes 2, 6, and 10. The z axis points out of the figure[2]. The octahedrons in a truss structure are all parallel to each other. Therefore, the transformation between the coordinate frames of two octahedrons is a pure translation.

Some tetrahedrons have three nodes on the top plane and one node on the bottom plane. These are referred to as *tetrahedron-down* because they can be viewed as a pyramid pointing down. The other tetrahedrons, which have three nodes on the bottom plane and one node on the top plane, are referred to as *tetrahedron-up*. All tetrahedrons-up in a truss structure are parallel to each other, and all tetrahedrons-down are parallel to each other. Therefore, the transformation between the coordinate frames of two tetrahedrons-up (or two tetrahedrons-down) is a pure translation.

A tetrahedral truss structure can be represented by a graph in which the vertices correspond to volumetric units, and the edges correspond to "face-contacts" between adjacent units. Figure 8 shows a graph representation for the 102-strut truss structure shown in Figure 1.

The geometry of this graph parallels that of the truss structure. Because of the regularity of the structure, its graph representation constitutes an hexagonal grid. In addition, the hexagonal grid can be mapped into a rectangular grid as shown in Figure 8, where the lines and columns are labeled with their indices. Furthermore, a coordinate frame can be associated with the graph shown in Figure 8: the x axis points down, and the y axis points right.

There are three types of vertices, represented, respectively, by hexagons, triangles, and half-hexagons. Hexagon vertices correspond to octahedrons like the one shown in Figure 3a. Triangles correspond to tetrahedrons; triangles pointing down in the figure correspond to *tetrahedrons-down*, and triangles pointing up correspond to *tetrahedrons-up*.

Unlike regular graphs, in this representation the position and the orientation of the vertices are important. A coordinate frame is associated with the

[1]Pentahedrons not contained in any octahedron occur at the periphery of a structure. See Figure 8.

[2]Nodes 4, 8, and 12 have positive z coordinate.

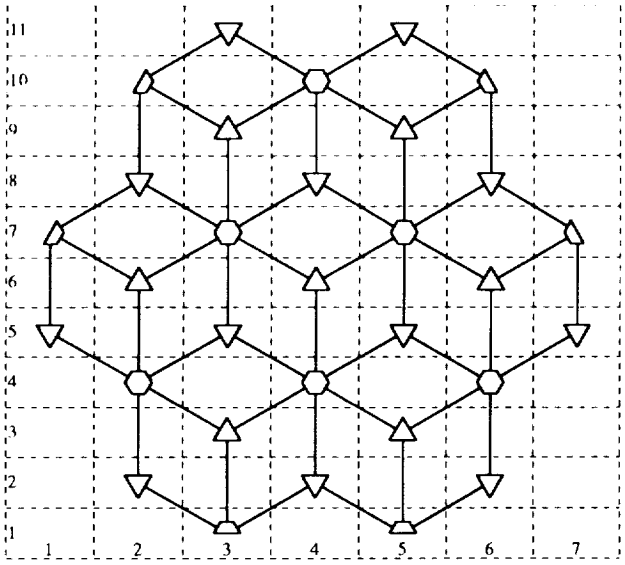


Figure 8: Graph representation for a tetrahedral truss structure with 102 struts and its mapping into a rectangular grid.

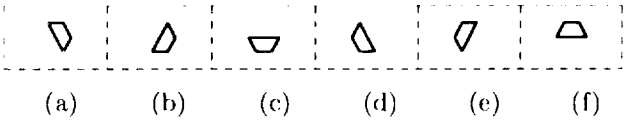


Figure 9: The six orientations in which half-hexagons may occur: (a) P2; (b) P4; (c) P6; (d) P8; (e) P10; (f) P12.

graph. Its axes are parallel to those of the frames associated with the octahedrons. Each vertex is oriented as its unit's parallel projection on the bottom plane of the structure.

The half-hexagons correspond to pentahedrons such as those shown in Figures 3b, 4a, and 4b. The half-hexagons are used only when there is no octahedron that includes the corresponding pentahedron. Since there are six pentahedrons embedded in each octahedron, there are six orientations in which the half-hexagons may occur, and they are shown in Figure 9.

The edges in the graph representation of a truss structure correspond to those "faces" that are common to two adjacent volumetric units, that is, those sets of three struts that "belong" to both volumetric units.

The mapping of the graph representation into a rectangular grid gives rise to a data-structure for a computer implementation: a two-dimensional array in which each element may contain information about one building block of the truss structure. The indices

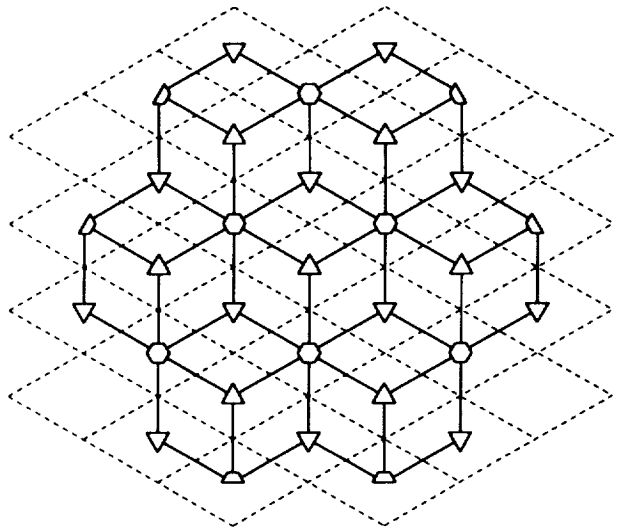


Figure 10: The mapping of the graph representation of tetrahedral truss structures into a quadratic (not rectangular) grid.

of the array element indicate the position of the building block.

The edges in the graph shown in Figure 8 are only implicitly encoded into the two-dimensional array data structure. In addition, the contacts between units that share only one strut, or only one node, are also implicitly encoded into the array. For example, a tetrahedron-up at cell $(i\ ,\ j)$ (i.e. line i, column j) shares one strut with the tetrahedron-down at cell $(i + 2\ ,\ j)$, another strut with the tetrahedron-down at cell $(i - 1\ ,\ j - 1)$, and another strut with the tetrahedron-down at cell $(i - 1\ ,\ j + 1)$. As another example, an octahedron at cell $(i\ ,\ j)$ shares one node with the tetrahedron at cell $(i + 2\ ,\ j - 2)$.

It should be pointed out that Figure 8 shows one mapping from the graph representation of tetrahedral truss structure, which is an hexagonal grid, into a rectangular grid. That mapping is probably the most direct, but it leaves a number of empty cells. In a computer implementation, if the available storage space is scarce, it is straightforward to devise other mappings from the hexagonal grid into a quadratic grid, which may not be rectangular. Figure 10 shows another mapping from the graph representation of truss structures into a quadratic grid. Unlike the one shown in Figure 8, the mapping shown in Figure 10 does not leave empty cells.

As it will become clear in the following subsections, this graph representation of truss structures allows an assembly planner to exploit the regularity in which the parts are joined to improve its planning efficiency. This improvement is due, in part, to the encoding of the geometry of the truss structure in the topology of

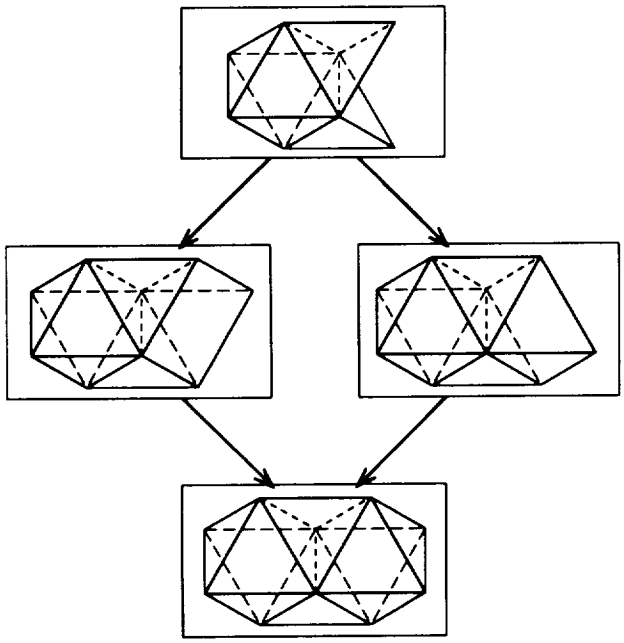


Figure 11: A portion of the directed graph of assembly sequences, which is also shown in Figure 12. The vertices have been labeled by the top view of the partial truss structure at each state of the assembly process.

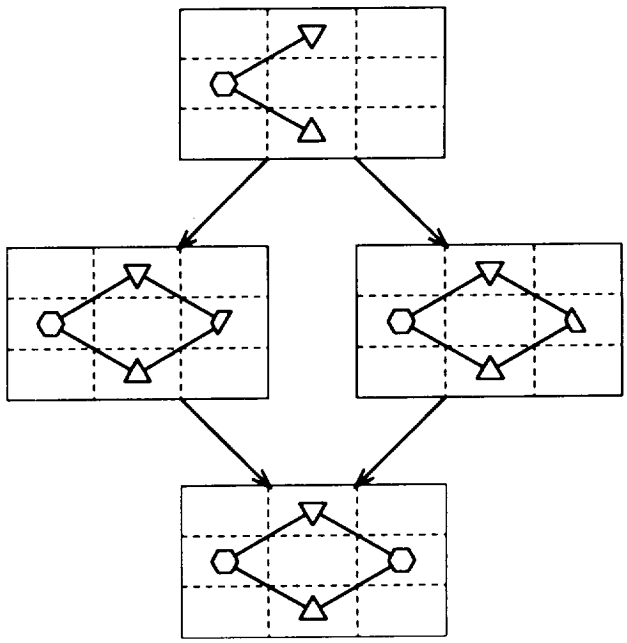


Figure 12: A portion of the directed graph of assembly sequences, which is also shown in Figure 11. The vertices have been labeled by the graph representation of the partial truss structure at each state of the assembly process.

the graph. Moreover, the graph representation also allows the planner to take advantage of the multiple hierarchies that exist in tetrahedral truss structures. The decision of which hierarchy to choose does not have to be made until it is needed. Being able to delay the selection of the hierarchy, the planner will have more information available to decide which hierarchy is more advantageous, and therefore will be able to make a better choice.

4.2 Control strategy

Several methodologies for representing assembly sequences have been utilized [7], including representations based on directed graphs and AND/OR graphs.

As mentioned in section 4, it is preferred to complete the assembly of a tetrahedral or pentahedral unit before beginning the assembly of another unit [11]. Therefore, the assembly task can be redefined as the assembly of one tetrahedron or one pentahedron. In this definition, each assembly task consists of a sequence of subtasks, each being the assembly of one strut.

Since each assembly task is the addition of exactly one volumetric unit, both the directed graph and the AND/OR graph will have the same size. The directed graph representation has been used in this work because it is simpler and easier to understand and implement. The vertices in this directed graph correspond to the states of the assembly process that can be characterized by the description of the substructure already assembled. The edges in this directed graph represent the assembly tasks, each corresponding to the addition of one volumetric unit.

Figures 11 and 12 show a portion of the directed graph of assembly sequences. In Figure 11 the vertices have been labeled by the top view of the partial truss structure at each state of the assembly process. This labeling is better for displaying the assembly sequences for humans. In Figure 12 the vertices have been labeled by the graph representation of the partial truss structure at each state of the assembly process. This labeling reflects more closely the computer internal representation of the assembly sequences. In both Figures, the vertex at the top corresponds to a state in which one octahedron and two tetrahedrons are already assembled. The two vertices in the middle corresponds to states in which an additional pentahedron is already assembled. In the left vertex, the additional pentahedron is P10, and in the right vertex, the additional pentahedron is P8. The vertex at the bottom corresponds to a state in which two octahedrons and two tetrahedrons are already assembled.

Figure 12 also illustrates the advantage of using the

multihierarchical representation of tetrahedral truss structures introduced in section 4.1. Because the building blocks are tetrahedrons and octahedrons, it is possible to generate sequences that use different sets of pentahedrons as assembly tasks. As pointed out above, the additional pentahedron in the left middle vertex is not the same as the one in the right middle vertex. By using the representation in Figure 8, the three possibilities in which an octahedron can be subdivided can be considered. In the scenario described in section 2, the two possibilities corresponding to the subdivision of the right octahedron into P6 and P12 are not considered valid. If the structure had been viewed as a composition of pentahedrons and tetrahedrons, only one alternative would be considered.

Each assembly sequence corresponds to a path in the directed graph of assembly sequences, starting in the vertex that has no label (i.e., no strut has been assembled) and ending in the vertex that is labeled by the whole truss structure. By construction, the directed graph of assembly sequences has no cycle. A measure that reflects the quality of an assembly sequence can be computed by assigning costs to the vertices (i.e., the states of the assembly process) and to the edges (i.e., the assembly tasks). The cost of a path p can be defined recursively as:

$$cost(p) = \begin{cases} C_S(s_p) & \text{if the path has only one node} \\ C_S(s_p) + C_T(t_p) + cost(r_p) & \text{otherwise} \end{cases}$$

where s_p is the initial vertex (state) of p, t_p is the initial edge (task) of p, and r_p is the tail of p, that is, what is left of p after s_p and t_p are removed. The function C_S gives an assessment of the quality of a state of the assembly process. Better (e.g., more stable) states correspond to smaller values of C_S. The function C_T gives an assessment of the quality of a task in the assembly process. Better (e.g., less complex or less time consuming) tasks correspond to smaller values of C_S.

The directed graph representation of assembly sequences and its associated cost function allow both backtracking and graph search control regimes [10] to be implemented. The construction of the assembly sequence can proceed in backward or forward fashion. The former is easier to understand while the latter may be more efficient, since it avoids dead-end states. Subsection 4.4 describes the current implementation.

4.3 Production rules

The global database introduced in subsection 4.1 reflects the state of the truss structure at each point of the assembly process. The production rules that are introduced in this subsection contain the conditions for the execution of an assembly task and the changes that occur in the state of the truss structure when that task is executed. In the operation of the planning system, whenever a production rule is applied, the global database must be updated to reflect the changes in the state of the truss structure.

- Precondition:
  1. Cell (i, j) currently contains a pentahedron Pk.
  2. Goal is one octahedron in cell (i, j).
  3. Any cell (x, y) for which $L(x, y, i, j, k) > 0$ is empty where $L(x, y, i, j, k) = \alpha(k) \cdot x + \beta(k) \cdot y + \gamma(i, j, k)$.
- Effect:
  1. Adjust the angle of the truss structure and the x–y position of the robot arm acording to the position of cell (i, j).
  2. Install pentahedron Pk' in cell (i, j) where $k' = \text{rem}(6 + k, 12)$.

Figure 13: Production rule example. See Table 1.

The simplest way to introduce the production rules is by an example. Figure 13 shows one production rule. It corresponds to the assembly task that finishes up one octahedron, starting with one of its pentrahedron halves already assembled. If the pentahedron already assembled is Pk, the pentahedron that will complete the octahedron is Pk' where $k' = \text{rem}(6 + k, 12)$.

The first two preconditions simply verify that the goal is an octahedron in a cell (i, j) that currently has a pentahedron. The third precondition verifies that no collision will occur between the truss structure and the carriage where the base of the robot is mounted. It requires that all cells on the side of the line $L(x, y, i, j, k) = 0$ where Pk' is must be empty. Figure 14 shows a state in which the preconditions of the production rule in Figure 13 are satisfied for cell $(7, 5)$ and $k = 10$.

The effect of this production rule is the installation of pentahedron Pk' in cell (i, j). This can be accomplished by using a precompiled sequence of subtasks, each of which is the addition of one strut. Since the base of the pentahedron is already in place, only four struts must added. This subsequence of tasks, each of which includes the motions of the robot arm, is independent of the position of the cell (i, j). Of course, the positions of the carriage and of the base, as well as the angle of the structure, must be adjusted according to the position of cell (i, j).

For each possible geometric configuration that a cell can take, there is a production rule similar to the

Table 1: Coefficients of $L(x, y, i, j, k)$ in the production rule example shown in Figure 13.

k	$\alpha(k)$	$\beta(k)$	$\gamma(i,j,k)$
2	-1	-3	$(i+3j)$
4	1	-3	$(3j-i)$
6	1	0	$-i$
8	1	3	$-(i+3j)$
10	-1	3	$(3j-i)$
12	-1	0	i

one in Figure 13. Since there are only a few geometric configurations, the total number of production rules is small.

4.4 Current implementation

The current implementation is an interactive production system that uses a backtracking control scheme. The assembly sequences are generated in a forward fashion. The first unit to be assembled is given.

At each step, a menu containing all the subunits that can be assembled next is displayed for the user. These options are obtained by testing the preconditions of the production rules. The alternatives in the menu are ranked according to the system's preference criterion. The user may accept the system's choice for the next subunit or may select another among those that are feasible. A graphical display of the truss structure allows the user to visualize the available options. At any point, the user can force the system to backtrack and to "undo" one or more assembly tasks.

This interactive production system exploits the strengths of humans and computers. Computers are better at guaranteeing that the sequence is correct and that no option is overlooked. Humans are better at assessing the quality of an assembly sequence.

The cost function that is used is a function of the translation of the carriage, the translation of the base, and the rotation of the structure. The shorter those motions, the lower the cost function. The task for which the cost function is minimal has the highest preference. Other cost functions are being investigated, and one of the goals of this project is to find good cost functions and their corresponding heuristic estimations.

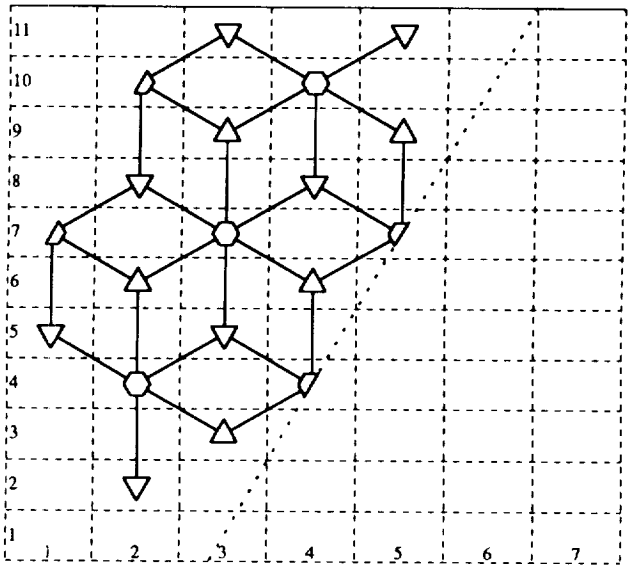


Figure 14: A state in which the preconditions of the production rule in Figure 13 are satisfied for cell $(7, 5)$ and $k = 10$.

In each assembly task, a number of struts are assembled. For example, in the tasks corresponding to the effect of the production rule shown in Figure 13, four struts are assembled. By properly positioning the carriage and the base of the robot, the arm motions to install a given strut is the same regardless of the position of the octahedron that is being completed. In the current implementation, these motions were taught. Each production rule is associated with the paths to install the struts of its corresponding subunit. Therefore, the output of the planning system includes, for each strut, the positions of the carriage and the base of the robot, the angle of the truss structure, and the specific arm motion to be used.

5 Conclusion

This paper brought about a clear understanding of the regularity of the tetrahedral truss structures and their multiple hierarchies. Unlike electromechanical and electronic devices, tetrahedral truss structures can be represented by a graph whose topology corresponds to the geometry of the parts. This representation captures the regularity of the truss structure as well as all its hierarchies. It consists of an hexagonal grid that can be mapped into a two-dimensional array data structure. The relationships between units are implicitly encoded by the indexes of their corresponding cells in the array.

Using this representation and its associated data structure, a simple reasoning is sufficient to decide

whether or not a candidate assembly task is feasible. Furthermore, the choice between hierarchies can be made as the plan is generated, thus allowing a better selection than if the choice were made in advance.

A prototype planning system that uses the production system paradigm has been implemented. The global database is the hexagonal grid representation of tetrahedral truss structures. There is one production rule for each possible configuration that a cell can take. Since there are only a few geometric configurations the total number of production rules is small. A directed graph representation of assembly sequences allows the use of both graph search and backtracking control strategies. The prototype uses a backtracking scheme.

This current implementation is interactive and exploits the strengths of humans and computers. Computers are better at guaranteeing that the sequence is correct and that no option is overlooked. Humans are better at assessing the quality of an assembly sequence. For the structure shown in Figure 1, the system generated an assembly sequence that significantly reduces the amount of rotation when compared to a sequence generated by hand. Future work will focus on cost functions and heuristic evaluations aimed at making the system fully autonomous.

Acknowledgments

The author would like to thank R. S. Desai for support of this research, and M. D. Rhodes and R. W. Will for valuable feedback on the planning system and for kindly providing Figure 2.

References

[1] D. F. Baldwin. *Algorithmic Methods and Software Tools for the Generation of Mechanical Assembly Sequences.* Master of science thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, February 1990. Also published as report CSDL-T-1040, The Charles Stark Draper Laboratory.

[2] A. Bourjault. *Contribution a une Approche Méthodologique de L'Assemblage Automatisé: Elaboration Automatique des Séquences Opératoires.* Thèse d'état, Université de Franche-Comté, Besançon, France, November 1984.

[3] T. L. De Fazio and D. E. Whitney. Simplified Generation of All Mechanical Assembly Sequences. *IEEE Journal of Robotics and Automation*, RA-3(6):640–658, December 1987. Corrections *ibid* RA-4(6):705–708, December 1988.

[4] J. M. Henrioud. *Contribution a la Conceptualisation de l'Assemblage Automatisé: Nouvelle Approche en vue de Détermination des Processus d'Assemblage.* Thèse d'état, Université de Franche-Comté, Besançon, France, December 1989.

[5] L. S. Homem de Mello. *Task Sequence Planning for Robotic Assembly.* PhD thesis, Carnegie Mellon University, May 1989.

[6] L. S. Homem de Mello and S. Lee, editors. *Computer-Aided Mechanical Assembly Planning.* Kluwer Academic Publishers, 1991.

[7] L. S. Homem de Mello and A. C. Sanderson. Representations of Mechanical Assembly Sequences. *IEEE Transactions on Robotics and Automation*, 7(2):211–227, April 1991.

[8] S. Lee and Y. G. Shin. Assembly Planning Based on Subassembly Extraction. In *Proc. IEEE Int. Conf. on Robotics and Automation.* IEEE Computer Society Press, May 1990.

[9] R. K. Mathur and A. C. Sanderson. A Hierarchical Planner for Space Truss Assembly. *Cooperative Intelligent Robotics in Space*, Rui J. deFigueiredo, William E. Stoney, Editors, Proc. SPIE 1387, 47–57, 1991.

[10] N. J. Nilsson. *Principles of Artificial Intelligence.* Tioga, 1980.

[11] M. D. Rhodes. Guidelines for Development of Truss Assembly Scenario. Unpublished technical notes, March 1990.

[12] M. D. Rhodes and R. W. Will. Automated Assembly of Large Space Structures. In *41st Int. Astronautical Congress*, October 1990.

[13] M. D. Rhodes, R. W. Will, and M. A. Wise. A Telerobotic System for Automated Assembly of Large Space Structures. NASA Technical Memorandum 101518, Langley Research Center, Hampton, VA, March 1989.

[14] S. Srikanth and J. U. Turner. Toward a Unified Representation of Mechanical Assemblies. *Engineering with Computers*, 6:103–112, 1990.

[15] R. W. Will and M. D. Rhodes. An Automated Assembly System for Large Space Structures. *Cooperative Intelligent Robotics in Space*, Rui J. deFigueiredo, William E. Stoney, Editors, Proc. SPIE 1387, 60–71, 1991.

[16] R. Wilson and J. F. Rit. Maintaining Geometric Dependencies in an Assembly Planner. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 890–895. IEEE Computer Society Press, May 1990.

SLS-PLAN-IT: A Knowledge-Based Blackboard Scheduling System for Spacelab Life Sciences Missions

Cheng-Yan Kao (+)
Dept. of Computer Science
& Information Engineering
National Taiwan University
Taipei, Taiwan 107

Tel: 886-2-363-0231 ext. 3231
Fax: 886-2-362-8167

&

Seok-Hua Lee (*)
GE Government Services
General Electric Corporation
1050 Bay Area Blvd
Houston, TX 77058, USA

Tel: (713) 488-9005
Fax: (713) 488-1092

ABSTRACT

The Mission Integration Office (MIO) of GE Government Services was responsible for generating and updating the crew activity plan and resource assignments for the Spacelab Life Science SLS-1 mission for NASA. The nine-day SLS-1 shuttle mission was launched on June 5, 1991.

The Spacelab mission planning was an overconstrained domain. There were over fifty resources and several hundred activities with several thousand steps to be scheduled in the SLS-1 mission. This is an NP-hard problem. The primary scheduling tool in use during the SLS-1 planning phase was the operations research (OR) based, tabular form Experiment Scheduling System (ESS) developed by Marshall Space Flight Center (MSFC).

PLAN-IT is an artificial intelligence (AI) based interactive graphic timeline editor for ESS developed by Jet Propulsion Laboratory (JPL). We have enhanced the PLAN-IT software for use in the scheduling of Spacelab experiments to support the Spacelab Life Science missions. The enhanced software SLS-PLAN-IT System was used to support the real time reactive scheduling task during the SLS-1 mission. This software will be further enhanced before the SLS-2 mission and is expected to completely replace the ESS currently in use in MIO in the SLS-3 time frame.

SLS-PLAN-IT is a frame-based blackboard scheduling shell which, from scheduling input, creates resource-requiring event-duration-objects and resource-usage-duration-objects. The blackboard structure is to keep track of the effects of event-duration-objects on the resource-usage-duration-objects. The constraints are propagated automatically for conflict resolution. Various scheduling heuristics are coded in procedural form and can be invoked any time at the user's request. The timeline entries can be manipulated by the mouse to support the scheduling task. This paper describes the system architecture and what we have learned with the SLS-PLAN-IT project.

(+) The first author was involved in this project when he was an employee of GE Government Services, Houston, Texas.

(*) All correspondence should be sent to the second author.

Introduction:

The Mission Integration Office (MIO) of GE Government Services was responsible for generating and updating the crew activity plan and resource assignments for the Spacelab Life Science SLS-1 mission for NASA. The nine-day SLS-1 shuttle mission was launched on June 5, 1991. The primary scheduling tool in use during the SLS-1 planning phase was the Experiment Scheduling System (ESS) developed by Marshall Space Flight Center (MSFC). The ESS software is hosted on a VAX computer. It has evolved over the past ten years into a FORTRAN program with 100,000 lines of FORTRAN code. However, it is very time-consuming in using ESS to update the crew activity timeline for the SLS missions. A joint effort between MSFC and Jet Propulsion Laboratory (JPL) of NASA went on for four years to develop an AI-based companion interactive graphic timeline editor, called PLAN-IT (shorthand for Plan-Integrated Timelines). PLAN-IT is a frame-based functional timeline manager. The objective was to enable the timeline engineers to explore scheduling options, recognize scheduling opportunities and thereby include additional or better-arranged activity into a schedule. The origin of PLAN-IT can be traced back to the AI planner DEVISER system of Vere (Ref. 13). Unfortunately, this joint effort of MSFC and JPL was terminated in October, 1988, and the PLAN-IT scheduling system was left unused in MSFC since then. Three main reasons for not using PLAN-IT were: (1) the integration of PLAN-IT into the ESS was poor, (2) the response time of PLAN-IT was unacceptable in certain cases, and (3) the resistance from the ESS developer and user communities was strong.

MIO of GE Government Services obtained the original PLAN-IT source code from JPL in 1988. We have enhanced the software for use in the scheduling of Spacelab experiments to support the Spacelab Life Science missions. The enhanced software SLS-PLAN-IT Scheduling System was used to support the real time reactive scheduling task during the SLS-1 mission. This software will be further enhanced for the SLS-2 mission, and is expected to completely replace the ESS Flight Planning System (ESS/FPS) currently in use by the MIO in the SLS-3 time frame. The SLS-PLAN-IT is currently hosted on the TI Micro-Explorer Lisp machine and will be ported to the SUN workstation under the Common Lisp Object System (CLOS) environment.

The project objective of SLS-PLAN-IT is to provide an intelligent scheduling tool that will allow the timeline engineers of the Payload Activity Planning team to interactively update an ESS generated timeline in a way that is time and cost effective. SLS-PLAN-IT is a decision support tool. Its purpose is to aid an expert human scheduler not only with effective graphics and a menu-driven interface, but also with natural problem presentation.

A vital feature of SLS-PLAN-IT is its resource timelines, which are similar to timelines normally in use. The timeline display shows the scheduler the conflicts in a trial sequence so

that the sequence can be modified and improved immediately. The sequence can be modified and the strategy can be directed while a strategy is running. There are several advantages to this approach. It allows the user to understand easily what is happening. The trial sequence is displayed directly on the screen. As the sequence changes incrementally, the user can quickly grasp what is happening and interact with the scheduling process. He can focus on some aspect of scheduling without being concerned of the other constraints. This feature also makes it easier for the user to capture expert advice.

There also exist controls that allow the user to focus on a strategy. For example, the user can disable some of the resource timelines so that the strategies will only consider a subset of the resources. After the basic schedule is laid out, additional resources can be evaluated. Another control approach is to tell SLS-PLAN-IT to work with certain types of activities or to consider moving activities within a user-defined window. The effect of SLS-PLAN-IT's strategies can be reduced by freezing some activities since only the user can move frozen activities. One of the more effective control approaches allows the user to select a single activity to which a strategy can be applied so that he can ask SLS-PLAN-IT if there is a better place in the sequence for this activity. This feature provides the scheduler with a "smart" sequence editor.

The goal of SLS-PLAN-IT is to achieve a blend of human and machine expertise. SLS-PLAN-IT initially produces preliminary layouts. After political decisions have been made, they will be reflected in the schedule. The operator can direct SLS-PLAN-IT to make minor changes in the sequence, or he can control the strategies. Finally, the operator can use SLS-PLAN-IT as an editor to verify that certain constraints have not been violated.

In the following sections, we will first describe the problem domain, review the relevant literature, then give a detailed description of the system architecture, report the current status and enhancement plan of the project, and finally discuss what we have learned from the project.

The Problem Domain:

Mission planning schedules are composed of three types of element: activities, resources, and constraints. Activities are the events in a schedule. They can either have durations (like experiment steps) or be point events (like a space shuttle launch). Activities consume, create, or replenish resources. Activities also have inter-relationships that are often expressed as precedence relationships or concurrency/non-concurrency of activities. Resources can be associated with one activity, a group of activities, or all activities. There are activity-specific resources, e.g., equipment associated with an experiment, and pool-resources, e.g., electrical power.

The Spacelab mission planning is an overconstrained domain.

In past Spacelab or Skylab missions, low priority experiments were occasionally bumped to achieve more important goals. Therefore, the mission planners must be able to relax or even ignore certain constraints in order to get an acceptable schedule.

In our timeline engineers' terminology, a performance is an execution of an experiment, and a step is an activity of an experiment. The experiments are then modeled by the constraints of the steps involved and the constraints of the performances. The constraints imposed for the Spacelab missions can be categorized into time constraints and resource constraints. The time constraints include performance time window, maximum and minimum performance duration, maximum and minimum performance delay, maximum and minimum step duration, maximum and minimum step delay, concurrency and non-concurrency of steps, and target or attitude opportunities. The resource constraints include equipment, nondepletable resources, depletable resources, resource carry-through, crew selection, crew lock-in, crew monitoring, and the requirements of balanced resource usage. In fact, this is an NP-hard scheduling problem.

Literature Review:

Bennington and McGinnis gave a survey of the past research in resource constrained project scheduling problems (Ref. 1). They demonstrated how to search for the optimal algorithm by three basic approaches: the first approach was to formulate the problem as an integer linear programming (ILP) problem, which can be solved by standard ILP techniques; the second approach was to directly employ some enumerative scheme for constructing an optimal schedule; and the third was to formulate the problem in terms of minimaximal paths in a disjunctive graph, which could be solved by network flow methods or implicit enumerations.

In spite of the progress in research, almost all researchers have agreed that the heuristic method is still the only viable solution technique for large-scale practical problems since the computing time would be prohibitively large if exact optimal procedures were used. Studies of the complexity of the resource constrained scheduling problems also draw lots of attention. Coffman showed that these problems were actually NP-hard (Ref. 3). Elmaghraby (Ref. 5) and Coffman (Ref. 3) contain excellent coverage of the recent results in resource constrained scheduling problems.

In recent years, the emergence of expert system technology has had a great impact on scheduling system design. Dhar and Ranganathan used the university course timetable scheduling problem as an example to contrast the advantages and disadvantages of AI approaches versus OR approaches (Ref. 4). They pointed out that the OR approaches had the following disadvantages:

1. Single objective limitations: The objective function used in OR formulation express one goal, but there are other goals

that the scheduling expert tries to satisfy.
2. Compiled knowledge limitations: Solutions are very sensitive to the coefficients of the objective function, and some default knowledge is difficult to incorporate into the coefficients.
3. Global optimization limitations: Global optimization essentially obscures the reasons for assignments and implies lack of explanation for its decisions.
4. Lack of support in making plan revisions: Plan revisions are inevitable, but it is very difficult for the decision maker to revise the schedule with minimum perturbation in OR approaches.

Jaap and Davis described an interesting review of the software development of ESS (Ref. 8). The ESS software hard-coded the scheduling rules in FORTRAN to handle the time constrains and the resource constraints. The scheduling core of ESS consisted of five modules: the bookkeeper for resource tracking, the checker for determining availability of resources, the loader to load the schedule, the trace listing as an explainer, and finally the selector to determine the ordering of scheduling the activities. Two methods, the random-order method and the preference-order method, were incorporated in ESS.

Boarnet documented the requirements of a scheduling expert system tool from NASA's point of view (Ref. 2). It was one of the best examples of the impact of expert system technology on the design of the scheduling system software. In the paper, Boarnet discussed the requirements of a scheduling expert system tool for Space Station Freedom mission planning applications. He pointed out that the scheduling tool should represent activities, resources, and constraints, with facilities to group those elements and to represent the time variance of the elements. The tool should support activity scheduling and job scheduling. Enumeration of alternatives with algorithms, hypothetical worlds, rule systems, and schedule hierarchies should be integrated into a powerful reasoning tool. The tool must support the procedural code that might be necessary either for procedure attachment or to control the scheduling techniques. The tool must support interactive scheduling with intelligence that can be interactive or automatic at the user's discretion, and with good human factors.

In the panel discussion on "AI-Based Schedulers in Manufacturing Practice" held in IJCAI-1989, Detroit, USA, Sidhu (Ref. 10) pointed out that the most common mistakes in building intelligent scheduling system include:
1. Inadequate analysis of dominant domain characteristics, especially when prepackaged scheduling tools are used.
2. Inappropriate reliance on locally greedy strategies. Because most scheduling problems are fairly complex, they are often simplified by using simple local dispatching rules.
3. Misuse of shallow expert knowledge: Human schedulers always over-simplify the constraints, or misrepresent situation-dependent knowledge as general-purpose knowledge.

The special issue of AI magazine, January 1991, contains a report of the workshop, "Issues in the Design of AI-Based Schedulers", by Kempf et al. (Ref. 9). The issues covered in the workshop included expert vs. deep vs. interactive schedulers, integrating predictive and reactive decision-making, maintaining convenient schedule descriptions, and some other advanced topics like learning and benchmarks. Several points expressed by the participants are very interesting and representative:

1. Fully automated schedulers are not as desirable as interactive schedulers because the man and the machine bring complementary skills to the scheduling task.
2. Many deployed scheduling systems contain only a small amount of AI. Successful systems can be dominated by other issues such as the user interface, database connections, and real-time data collection.
3. One strong point for interactive methods is that they allow humans to build schedules by methods that they naturally use but are hard to represent, and allow humans to guide the search.
4. Integration of predictive and reactive scheduling components is important. A blackboard-style scheduling system architecture may be appropriate.
5. Optimization is an ill-conceived objective for scheduling. It is hard to define, and factory operations are unpredictable.

Fox and Smith proposed a knowledge-based system for factory scheduling called ISIS (Ref. 6). The central idea of ISIS is that schedule construction can be cast as a constraint-directed activity that is influenced by all relevant scheduling knowledge. In the paper, they pointed out that given the conflicting nature of the domain's constraints, the problem differs from typical constraint satisfaction problems, and one cannot rely solely on propagation techniques to arrive at an acceptable solution. Rather, constraints must be selectively relaxed and the problem-solving strategy must be one that finds a solution that best satisfies the constraints. This implies that the constraints must serve to discriminate among alternative hypotheses as well as to restrict the number of hypotheses generated. The design of ISIS focused on two issues:

1. Construction of knowledge representation that captures the requisite knowledge of the job shop environment and its constraints to support constraint-directed search, and
2. Development of a search architecture capable of exploiting this constraint knowledge to effectively control the combinatorics of the underlying search space.

In constructing a job shop schedule, ISIS conducts a hierarchical multi-level constraint-directed search in the space of all possible schedules. The different levels of the search provide multiple abstractions of the scheduling problem, each a function of the specific types of constraints that are considered at that level. Control generally flows in a top down fashion, and communication between levels is accomplished via the exchange of constraints.

Syswerda and Palmucci presented the construction of a genetic algorithm based optimizer for a resource scheduling application (Ref. 12). Genetic algorithms (GA) use Darwin's fitness-for-survival principle to do function optimization. The optimizer described in the paper is a combination of local expert search and global search provided by a genetic algorithm. The issues involved in the construction of a GA-based scheduler include:

1. how to represent the schedule as a bit-string used in GA,
2. how to isolate the details of the problem from the GA.

They also pointed out that the system must be able to combine manual scheduling of special cases with automatic scheduling based on more general criteria. Manual scheduling is accomplished by the use of an intelligent graphical interface. The interface is intelligent in that it understands all the well-defined constraints of the scheduling problem, and advises the user about where to place tasks while disallowing the construction of illegal schedules. We have similar graphical user interface in SLS-PLAN-IT.

SLS-PLAN-IT's System Architecture:

SLS-PLAN-IT's approach to problem solving relies on three highly interactive elements: a model builder to construct activity and resource models, a user interface that takes into consideration what the user needs to know and how he controls or directs the scheduling process, and the scheduling strategies.

Hayes-Roth (Ref. 7) used a blackboard model to implement their opportunistic strategy, planning both top-down and bottom-up. Smith et al (Ref. 11) reported an extension of ISIS to OPIS (the Opportunistic Intelligent Scheduler), which was implemented with a blackboard style architecture. These knowledge sources had implemented alternate scheduling strategies that extended and revised a global set of scheduling hypotheses. Smith et al. reported better performance than that of ISIS with the multiperspective scheduling approach. These blackboard models have beed adopted by SLS-PLAN-IT scheduling system.

The blackboard structure is a global, hierarchical data structure partitioned to represent the problem domain as a hierarchy of analysis levels. Each level consists of nodes that are objects in the system implemented as frame structures. The nodes are integrated by links, where a node in the hierarchical structure represents an aggregation of lower level nodes. Thus, the blackboard can be structured as an undirected graph of nodes. However, one can place nodes without links on the blackboard. During problem solving, partial schedules begin to grow on the blackboard. The higher levels represent abstract decisions made about the general pattern of the mission schedule, while the lower levels represent decisions made about the specific details of the schedule. The relationships of the nodes are either specified by the model builder prior to the scheduling sessions, or specified via the mouse by the user dynamically during the

manual-mode scheduling sessions. Thus, knowledge sources can create decisions that refine the schedule from the higher to the lower levels of the blackboard, growing the schedule in a top-down fashion. Alternatively, knowledge sources can create decisions about specific details of a schedule and incorporate those decisions into the whole schedule, growing the schedule in a bottom-up fashion. The knowledge sources are specialists that access the blackboard by creating nodes, modifying nodes, or modifying links between nodes. This allows a knowledge source to contribute information without knowing which other knowledge sources will be using the information. In SLS-PLAN-IT, the knowledge sources are implemented as scheduling strategies that can be triggered whenever a goal is posted or whenever data changes. The three main components of the system are described in detail in the following sections.

Model Builder

SLS-PLAN-IT uses a datatype specification modeling language to model the scheduling requirements of the mission. The purpose of having the modeling language is two-fold. Firstly, it is to simplify resource definitions programming so that classes of items already defined need not be re-coded by hand. Secondly, it is to insulate the resource-describer from having to know the exact order of resource definition commands that must be included in the source code.

In response to the users' request to remove the major obstacle that discourages the users from using SLS-PLAN-IT, a model builder is currently under development and will be included into SLS-PLAN-IT for the SLS2 mission.

The model builder will be able to construct activity models and resource models. An activity can be an experiment to be scheduled or an electrical storage to be discharged. Resources include the depletables, the non-depletables and the human resource. Examples include the power, the data rate, and the crew.

An activity model will include a series of individual steps to be performed in the experiment, the scheduling time ranges or time allotment, the resources to be used, and the constraints in scheduling. The steps in an activity may occur sequentially or concurrently, or they may overlap one another.

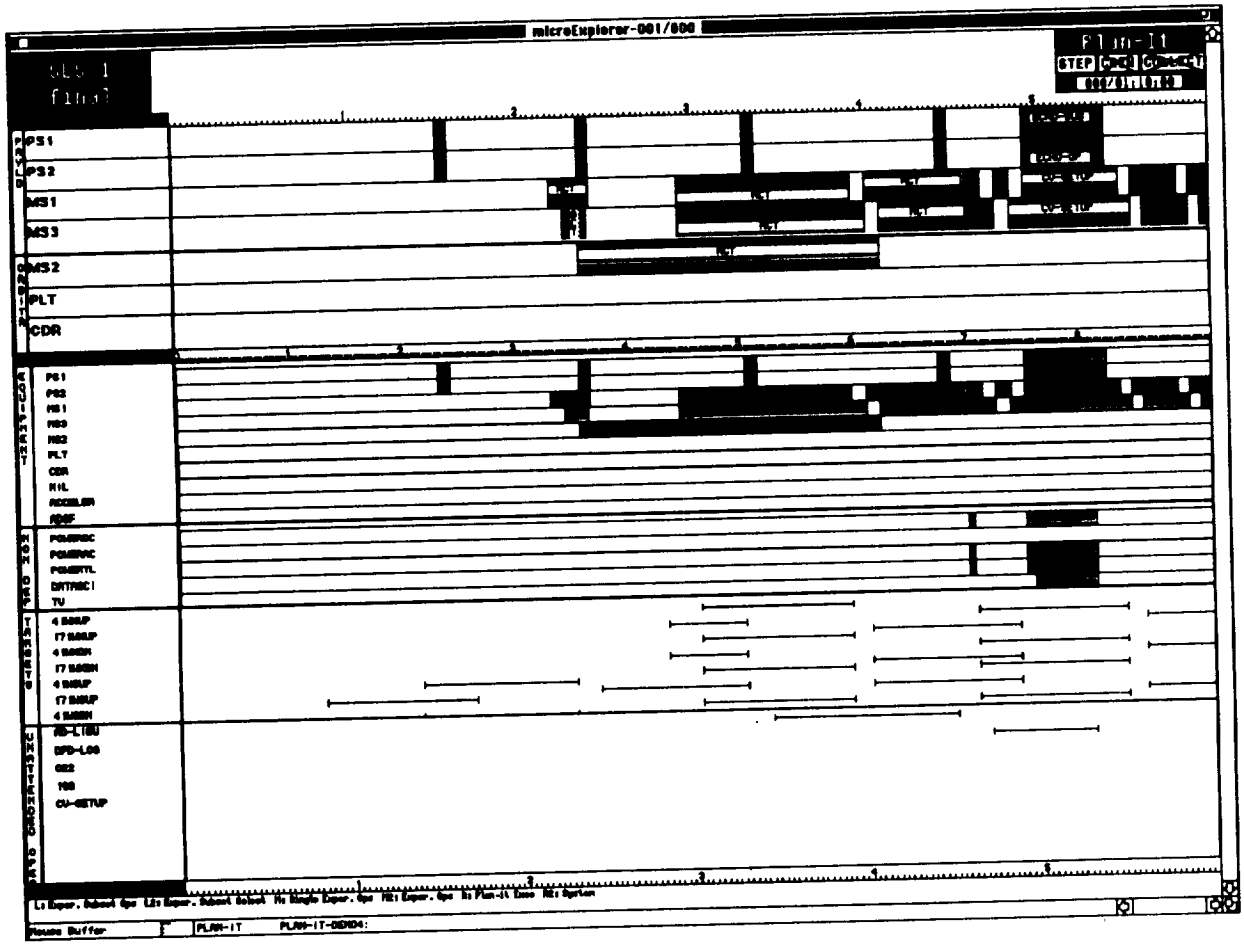
Resources are modeled as timelines that show how each resource is used or changed throughout the entire sequence. A resource model will include the availability of the resource in quantity and time, and the constraints in scheduling.

The activities and the resources interact with each other throughout the scheduling process. Whenever an activity or a step of an activity is changed, the resource timelines will be updated. Whenever the usage of any resource has exceeded its limits, a conflict will be detected. In this way, the resource

models will serve as safeguards against the misallocation of the resources.

User Interface

SLS-PLAN-IT's man-machine interface focuses on the graphical presentation of the resources and activities. Control of the program is through pop-up menus and mouse operations. The screen of SLS-PLAN-IT is divided into six sections (see Figure). The top section includes a status pane that displays operational messages of the program, or the status information of the mouse, or the detailed information related to the timeline interval over which the mouse is positioned. The next five sections are graphical displays of the experiment timelines, the equipment-resource timelines, the non-depletable-resource timelines, the target or attitude opportunity information window, and the unattended-operation timelines, one below another. The unattended-operations consume resources, but no crew were associated with them except for occasional monitoring. The details of an experiment can be edited interactively. The



Figure

resource timelines display white where there is no load, gray where loads exist with no resource conflict, and black to denote a conflicting area. By positioning the mouse over an activity, details of the activity will be displayed in the status pane to aid the user in editing. When the mouse is over a resource timeline, the status pane will display the amount of the resource being used and the activities involved. When the mouse is over a conflicted resource area, the status pane will display the activities that caused the conflict.

When the entire SpaceLab sequence of the mission is displayed, the screen is overwhelmed by the amount of detail. Therefore, SLS-PLAN-IT provides a zooming facility for the user to tailor the screen display to his/her need. The user can examine any portion of the timelines at any specified scale. Since SLS-PLAN-IT's display is interactive, the user can actually watch during the automatic mode what the scheduling strategy is doing as the experiments are being moved and modified. The impact on the resource timelines and on the experiments is directly and immediately shown to the user. At any point in the processing, the user can redirect SLS-PLAN-IT to focus on a different aspect of the sequence.

There are several modes of operation in SLS-PLAN-IT, from running without user interaction to user controlling the search or user manually scheduling the experiments. Therefore, the user can select the level of control over the mission timeline, and go back and forth among the various modes of operation.

An important feature of SLS-PLAN-IT is the explicit conflict representation on resource timeline. This is a natural representation for the expert user and thus made the interaction with the user more direct and simpler. Since the experiments are tracked explicitly on the Gantt-chart type timeline blackboards, the experiments could be scheduled in any random order.

The ability of an expert scheduler to intuitively grasp what the scheduling engine is trying to do is very important, as has been noticed by several researchers as a necessary condition of a successful scheduling system. SLS-PLAN-IT developers are well aware of this.

Scheduling Strategies

Besides the manual scheduling mode supported by the blackboard structure, the constraint propagation mechanism and the graphical user interface, SLS-PLAN-IT also supports automatic scheduling mode with various scheduling strategies. One of the fundamental ideas of SLS-PLAN-IT is that there is no single "correct" way to sequence. In fact, no single way is powerful enough to do the task in a reasonable time. Thus, SLS-PLAN-IT supports a number of scheduling strategies that can then be combined into a scheduling session.

In automatic mode, the system must be able to compare

different partial schedules and choose one to continue scheduling. This requirement is reflected in the structure of scheduling strategy. A strategy consists of three parts. The first part is a goodness measure that indicates whether one sequence is better than another. This measure can change from strategy to strategy. Typically, the goodness measure rates the total conflict on the resource lines. The second part is to select activities to be changed, which can simply be all the activities of certain types or the activities involved in the worst conflict. The third part is to suggest the actions to be taken such as to move, to modify, or to delete an activity. SLS-PLAN-IT makes small changes, one at a time, to improve a goodness rating.

Several strategies are currently implemented in SLS-PLAN-IT. These strategies together form a hill climber. A goodness rating will determine a topology for the search space. A strategy will change the schedule until it finds a local maximum in the topology of the search space. By selecting a different strategy, the topology of the space will be changed and the SLS-PLAN-IT will be able to continue improving the mission sequence.

SLS-PLAN-IT possesses meta-knowledge in the form of strategy modifiers. These modifiers restrict the search space of a strategy. An example of this is the restriction on the number of resources a strategy could consider. This particular modifier is based on the knowledge that, to the first order, a mission schedule is determined by a small subset of the total number of resources. Other modifiers force a strategy to only consider moving experiments to areas of the mission timeline that have little resource usage.

There seems to be no single best way in scheduling. The scheduling techniques depend on the particular project, the life point in the mission, and the current schedule. SLS-PLAN-IT is able to represent many different scheduling strategies. Flexibility in choosing a suitable scheduling strategy is the key to successful scheduling. The concept of scheduling strategy provides a natural hook of SLS-PLAN-IT system to any optimization technique. Given a goodness measure as the objective function, all the scheduling and sequencing techniques available from traditional operations research discipline or non-traditional combinatorial optimization approaches can be incorporated into SLS-PLAN-IT in the form of scheduling strategies.

Current Status of SLS-PLAN-IT:

MIO of GE Government Services obtained the original PLAN-IT source code from JPL in 1988. We have tailored the software for use in the scheduling of Spacelab experiments to support the Spacelab Life Science missions. Although the original PLAN-IT has the ability to perform very specialized strategies to resolve particular scheduling difficulties, the automatic mode that uses the above strategies is still not powerful enough to handle the overconstrained resource requirements of scheduling the Spacelab

mission timeline. MIO's current major concerns in SLS-PLAN-IT to support the SLS-2 mission include the graphical user interface and the automatic constraint propagation capability, which allow the user to modify a timeline by mouse operations. We will enhance the automatic scheduling capabilities for SLS-3.

A Spacelab mission timeline contains over fifty resources and hundreds of experiments. The timeline engineers can manually enter the initial schedule or use MSFC's tabular-form scheduler ESP to produce the initial schedule. During SLS-1 mission planning phase, the timeline engineers used SLS-PLAN-IT to maintain the schedule produced by ESP. After the schedule was modified by SLS-PLAN-IT, the modified schedule was transmitted to the Flight Planning System (FPS) for standard output plotting.

The SLS-PLAN-IT Scheduling System was used to support the Spacelab Life Sciences-1 (SLS-1) mission during the mission period from 6/05/91 to 6/14/91. This on-line real time usage of SLS-PLAN-IT during the mission demonstrated the strength of this scheduling system. The timeline engineers of the Payload Activity Planning (PAP) team confirmed that SLS-PLAN-IT is a flexible and useful scheduling tool that provides a real time reactive planning capability that the old scheduling system ESP/FPS lacks. For example, the timeline engineer had used SLS-PLAN-IT to reschedule the activities on flight day 9 due to short notice. The ESP required more time than available to do this kind of replanning. The SLS-PLAN-IT had won the user confidence and acceptance that were not in existence in the early stage of PLAN-IT development.

Feedbacks from SLS-PLAN-IT users during SLS-1 mission are:

1. The system gives visual display of experiment timeline with schedule conflicts indicated. Mission planning using SLS-PLAN-IT is much quicker than using ESS.
2. The mouse and menu-driven user-interface of SLS-PLAN-IT and the Gantt-chart like resource timeline are very convenient to support manual-mode scheduling of the mission. Minimum user training is needed for manual-mode scheduling if SLS-PLAN-IT is used, instead of the six months training time of ESS.
3. It is usable as a real-time reactive mission scheduler with prospects of increasing productivity of mission support staff and increasing science returns of the Spacelab experiments.
4. A quicker and more convenient model-builder is needed to support the SLS-2 mission. The integration of SLS-PLAN-IT with other flight planning software needs to be improved.

Enhancement:

The on-line real time usage of SLS-PLAN-IT aroused the user interests to further enhance the SLS-PLAN-IT software. The major enhancement requirements include:

1. rehosting SLS-PLAN-IT to a SUN platform to boost the operation speed and to allow better integration,
2. providing an intelligent model builder to enhance the model editing capability and shorten the modeling and planning time,

3. providing more options for automatic file creation and generation of operation output in current FPS format, which includes the information of ground tracking, attitude timeline, sun/shadow times and all other miscellaneous information on other FPS output.
4. additional capabilities to support the execute shift activities; the exact requirements are to be determined.

The direction we are taking is to completely replace the FPS system with SLS-PLAN-IT in the SLS-3 time period. With a very high level of user involvement, the SLS-PLAN-IT will evolve as a fully automated knowledge-based scheduling system with graphical user interface for space exploration.

In summary, the performance of SLS-PLAN-IT during the SLS-1 mission was very satisfactory. Recommendation for further enhancements of the software was made for the SLS-2 mission. It is expected that the SLS-PLAN-IT will completely replace the ESS currently in use by the MIO in the SLS-3 time frame.

Conclusion and lessons learned:

During the development of SLS-PLAN-IT, we have gained some useful experience in software engineering of an AI-based scheduling system that we would like to share with the community:

1. Quick response time is crucial in real time scheduling environment. One of the reasons that JPL's version of PLAN-IT was abandoned is that it did not have a model editor. It took an hour or more on Symbolics 3650 to incorporate the model from ESS. We improved the operation time to about ten minutes in the first version of SLS-PLAN-IT. Enhancements of the model editor to support incremental model editing are in progress.

2. Good integration of the AI scheduler with all other Flight Planning System (FPS) software is important because the purpose of SLS-PLAN-IT is for operational daily usage to support the mission.

3. Automatic shift of focus is difficult to achieve. There are many scheduling strategies available in the automatic mode of PLAN-IT. However, the users did not use them for the SLS-1 mission. One of the reasons is that the users do not fully comprehend the scheduling strategies. We have to better express the strategies to the users in more natural ways or the "automatic mode" will stay unused.

4. It is very important to allow the users to play "what-if" games during scheduling process and see why things happened. This is one of the reasons the MIO mission planners switched from using ESS to using SLS-PLAN-IT.

5. User-naturalness is the key to have a good user interface. For example, the automatic constraint propagation capability of SLS-PLAN-IT and the blackboard structure of the resource lines

are user-natural tools to support the above vital features.

6. The level of user involvement and expectation of SLS-PLAN-IT is very high in MIO over MSFC and JPL. In fact, the users always expect more than the developers can provide. We are driven by the users and the users are driven by the scheduling workloads they support.

7. The effects of a schedule change should be kept as local and as minor as possible. Minimum disruption of the schedule is sometimes more important than obtaining an optimal schedule.

Our experience with SLS-PLAN-IT reconfirms the observations made in the IJCAI workshop mentioned earlier. Fully automated schedulers are not as desirable as interactive schedulers because the man and the machine bring complementary skills to the scheduling task. Also, deployed scheduling systems contain only a small amount of AI. The issues of user interface, database connection, and real-time requirements dominate the system design and user acceptance of the scheduling system. The most important feature emphasized in SLS-PLAN-IT is the user-natural interface to cooperate with humans in changing perspective or focus level to support the opportunistic scheduling strategies. The various strategies employed in the automatic scheduler are attempts to simulate the opportunistic scheduling capability of the human.

Acknowledgment:

The authors would like to thank Mr. Michael Hollander and Mr. William C. Eggemeyer of Jet Propulsion Laboratory for providing us the source code of the original PLAN-IT software and giving us valuable advice during the software conversion period. We would also like to thank all the timeline engineers of MIO/GEGS, Houston, Texas, for giving us their requirements and user feedback concerning the SLS-PLAN-IT Scheduling System.

This work is performed by GE Government Services, Johnson Space Center, Houston Texas in support of the NASA Mission Management Office government contract NAS9-17884.

REFERENCE:

1. Bennington, G.E., & McGinnis, L.F. (1972). A Critique of Project Planning with Constrained Resources. Symposium on the Theory of Scheduling and its Application, Springer-Verlag, New York, 1973.

2. Boarnet, M.G. (1986). Requirements for a Scheduling Expert System Tool. NASA/JSC Mission Planning and Analysis Division, Internal Memorandum #FM7(86-66), April 1986.

3. Coffman Jr., E.G. (1976). Computer and Job Shop Scheduling Theory, John Wiley and Sons, Inc., New York, 1976.

4. Dhar, V., & Ranganathan, N. (1990). Integer Programming vs.

Expert Systems: An Experimental Comparison, CACM March 1990, pp. 323-336.

5. Elmaghraby, S.E. (1973). Symposium on the Theory of Scheduling and Its Applications, Springer-Verlag, Berlin 1973

6. Fox, M.S., & Smith, S.F. (1984). ISIS--A Knowledge-Based System for Factory Scheduling. Expert System, Vol 1, No. 1, July, 1984.

7. Hayes-Roth, B. (1985). A Blackboard Architecture for Control. Journal of Artificial Intelligence, Vol 26, pp 251-321.

8. Jaap, J., & Davis, E. (1986). Expert Scheduling for Spacelab Mission. Proceeding of Conference on Space Applications of Artificial Intelligence, Huntsville, AL, Nov 13-14, 1986.

9. Kempf, K., Pape, C., Smith, S.F., & Fox, B.R. (1991). Issues in the Design of AI-Based Schedulers: A Workshop Report. AI Magazine, Special Issue Jan. 1991. pp. 37-46.

10. Sidhu, S. (1989). Avoiding Typical Mistakes while Building Intelligent Scheduling System. Panel Discussion on AI-Based Schedulers in Manufacturing Practice, IJCAI-89, Detroit, Michigan, USA, August, 1989.

11. Smith, S.F., Fox, M.S., & Ow, P.S. (1986). Constructing and Maintaining Detailed Production Plans: Investigations into the Development of Knowledge-Based Factory Scheduling Systems. AAAI's AI Magazine, Vol. 7, No. 4, Fall 1986.

12. Syswerda, G., & Palmucci, J. (1991). The Application of Genetic Algorithms to Resource Scheduling. Proceeding of the Fourth International Conference on Genetic Algorithms (ICGA-1991), San Diego, CA, July 13-16, 1991.

13. Vere, (1981). Planning in Time: Windows and Durations for Activities and Goals. IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol. 5, No. 3, pp. 246-267, May, 1981.

Detecting Opportunities for Parallel Observations on the Hubble Space Telescope

Michael Lucks
Space Telescope Science Institute*
3700 San Martin Drive
Baltimore, MD 21218

Abstract

The presence of multiple scientific instruments aboard the Hubble Space Telescope provides opportunities for *parallel science*, i.e., the simultaneous use of different instruments for different observations. Determining whether candidate observations are suitable for parallel execution depends on numerous criteria (some involving quantitative tradeoffs) that may change frequently. This paper presents a knowledge based approach for constructing a scoring function to rank candidate pairs of observations for parallel science. In the Parallel Observation Matching System (POMS), spacecraft knowledge and schedulers' preferences are represented using a uniform set of mappings, or *knowledge functions*. Assessment of parallel science opportunities is achieved via composition of the knowledge functions in a prescribed manner. The knowledge representation, knowledge acquisition, and explanation facilities of the system are presented. The methodology is applicable to many other multiple-criteria assessment problems.

1. Introduction

Despite a well-known manufacturing flaw in its primary mirror, NASA's orbiting Hubble Space Telescope (HST) has produced images of unprecedented clarity since its launch in 1990 [Kin91]. Repair of the telescope's ability to resolve very faint objects is planned during a maintenance mission in 1993, and demands for observation time on the HST have remained high.

In order to maximize the efficient use of observation time, the Space Telescope Science Institute (STScI) has promoted the research and development of advanced methods for scheduling astronomical observations. Two software systems -- Spike and Transformation -- developed at STScI have applied artificial intelligence techniques toward this end. Transformation [Ger91] is a planning system for the grouping and ordering of observation tasks. Transformation converts observers' requests for spacecraft activities into hierarchical structures called *scheduling units* (SUs) containing multiple sequential tasks that may be subsequently treated as indivisible entities for scheduling purposes. Spike [Mil91, Joh90a, Mil88] is a knowledge-based system for long-range scheduling. Using *suitability functions* [Joh90b] to represent scheduling constraints and preferences, Spike determines week-long segments into which each SU may be scheduled. The output of Spike is later refined into a second-by-second calendar using the Science Planning Scheduling System (SPSS) [Tay91].

The presence of six scientific instruments aboard the HST provides opportunities for *parallel science*, i.e., the simultaneous use of different instruments to observe different targets. By overlapping multiple observations, this concept clearly has the potential to increase throughput. Parallel science is particularly useful for scheduling important exploratory surveys. Without it, such explorations consume considerable resources at the expense of many other shorter and more specific observations. If executed in parallel with other pre-scheduled activities, however, such endeavors may be undertaken at

* Operated by the Association of Universities for Research in Astronomy for the National Aeronautics and Space Administration

opportune times without excessive resource consumption. The parallel science effort at STScI has been a substantial undertaking, involving major extensions to several existing software systems, including Spike, Transformation, and SPSS.

To utilize parallel science, an observer must specifically request that an observation is to be conducted in parallel. It is then the responsibility of schedulers at STScI to find a corresponding non-parallel (or *primary*) observation somewhere on the HST calendar with which the parallel task may be matched. The large number of observations (on the order of 10^4 primaries and 10^3 parallels) and the wide diversity in their requirements makes this a formidable task. This paper describes the Parallel Observation Matching System (POMS), a knowledge-based advisory system embedded into Spike that assists schedulers in finding such matches between primaries and parallels. For each primary SU scheduled by Spike, POMS ranks available parallel SUs according to their compatibility with the requirements of the primary.

Although compatibility between a primary and a parallel depends on certain obvious factors such as instrument constraints (e.g., both tasks must not require the use of the same instrument) and pointing similarity (the two targets must be sufficiently close), it also involves many other more subtle criteria, some of which are quantitative in nature and introduce tradeoffs into the assessment. The advisory system must be able to represent and aggregate the effects of such criteria.

Another salient characteristic of the problem is that the match assessment criteria are likely to be vague, particularly in the early stages of the parallel science project, since parallel scheduling policies have not been firmly established and the full effects of the criteria are not yet well understood. Hence, knowledge is likely to be tentative, incomplete, and subject to frequent change. Ease of incremental extension and modification of the knowledge base is therefore crucial to the success of the system.

The knowledge representation scheme used in POMS permits the construction of a scoring function for ranking primary-parallel matches. The scoring function is built from modular units of knowledge about individual criteria and from a modifiable aggregation formula that is an explicit part of the knowledge base. The approach has been used previously in a very different application, i.e., a prototype advisory system for selecting mathematical software from numerical subroutine libraries [Luc92]. POMS is the first production level application of the technology. The approach extends existing representation media in its capabilities to express quantitative tradeoffs and complex interactions among multiple criteria [Luc90], while retaining the traditional advantages of expert systems for incremental modification and explanation.

POMS is implemented in the Common Lisp Object System (CLOS) programming language.

The remainder of the paper is organized as follows. Section 2 describes the architecture and high-level functionalities of the system. Section 3 introduces some terminology and notation required to describe the knowledge representation. The assessment criteria are described in Section 4. The knowledge representation, explanation, and knowledge acquisition facilities are presented in sections 5, 6, and 7, respectively. In section 8 we discuss the preliminary results of the POMS project.

2. System Overview

The architecture of POMS is depicted in Figure 1. The three major components of the system are: (1) the parallel database, (2) the parallel knowledge base, and (3) the parallel matcher.

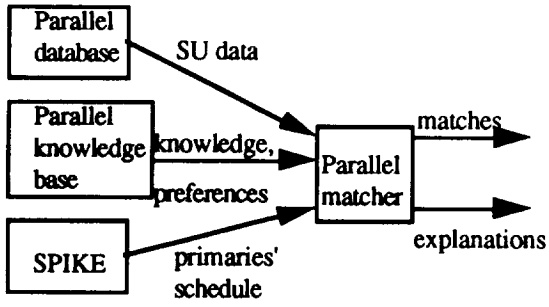


Figure 1. Architecture of POMS

The *parallel database* is a relational database that stores descriptive information about each primary and parallel (e.g., pointing, instrument, and timing requirements). This information is accessible to the matcher when needed at run time.

The *parallel knowledge base* contains the assessment criteria used to evaluate the compatibility between primaries and parallels. The knowledge base was initially constructed by the POMS developers (with input from systems engineers), however it may be modified by users (schedulers).

The *parallel matcher* is the POMS control mechanism. The matcher is invoked after the primaries have been scheduled by Spike to week-long segments, but before these primaries are delivered to SPSS for short-term scheduling refinement. The matcher evaluates the compatibility of each scheduled primary with each available parallel in the database, based on the descriptions of the SUs contained in the database and the scheduling knowledge/preferences contained in the knowledge base. Each primary-parallel pair is assigned a score, and the highest ranked matches for each primary are delivered to SPSS.

Upon request, the matcher also generates explanations of its advice for inspection by users and knowledge engineers.

3. Terminology and notation

The following notation and terminology are used in the subsequent description of POMS.

P is a set of primaries that have been scheduled for a specific week by Spike.

p is a primary SU in P.

Q is a set of parallel SUs to be matched with P.

q is a parallel SU in Q.

$R = \{r_1, r_2 ..., r_k\}$ is a set of *properties* that characterize primaries and/or parallels (target, time duration, instrument, etc.). Properties may be viewed as functions on $P \cup Q$, e.g.:

if r_1 = "primary-target", $r_1(p)$ returns a list of two elements containing the celestial longitude and latitude of the primary's target;

if r_2 = "parallel-instrument", $r_2(q)$ returns the name of the scientific instrument required by the parallel;

if r_3 = "primary-duration" $r_3(p)$ returns number of seconds allotted to the primary during which parallel science may be conducted;

if r_4 = "primary-week" $r_4(p)$ returns the week to which p has been scheduled.

To avoid ambiguity, properties that apply to both primaries and parallels (e.g., "target" or "instrument") are represented as two distinct properties ("primary-target," "parallel-instrument"). For notation, assume that properties 1,2,...,h apply to primaries, and properties h+1, h+2,..., k apply to parallels, i.e., $R = r_1, r_2, \ldots r_h, r_{h+1}, r_{h+2}, \ldots r_k$. Property values either exist explicitly in the parallel database (e.g., the primary or parallel instrument), or are determined by the Spike scheduler (e.g., the week to which a primary has been scheduled).

$F = \{f_1, f_2, ..., f_n\}$ is a set of evaluation criteria or *features* upon which the compatibility between a primary and a parallel is assessed. Examples of features for a primary p and a parallel q are: "pointing" (the targets of P and Q must be sufficiently close to one another), "instruments" (p and q must not both require the use of the same scientific instrument), and "timing" (p and q match best when they each require approximately the same amount of execution time). Note the distinction between features and properties, i.e., properties are characteristics of a primary or a parallel (e.g., the instrument used), while features are characteristics of a match (e.g., whether the instruments used by the primary and the parallel are the same). Property values are required for the evaluation of features (see below).

The *evaluation interval* $L = [L_l, L_h]$ is a sub-interval of the real numbers, where $L_l < L_h$. L_l is the lower bound of L and L_h is the upper bound. L is the range of the scoring function, as well as the range of some of the mappings used in the knowledge base. In the POMS knowledge base L = [0,1], although this restriction need not hold in general.

The *neutrality point* $L_N \in L$ indicates a "neutral" value in the evaluation interval. This reflects a "moderate" compatibility score (i.e., not particularly compatible, nor particularly incompatible). In POMS, $L_N = 0.5$.

The *scoring function* $H : P \times Q \rightarrow L$ evaluates the compatibility of any primary p with any parallel q. For all $p, p' \in P$ and $q, q' \in Q$, $H(p, q) > H(p', q')$ iff the match between p and q is considered better than the match between p' and q'. Since any p or q is described by properties, we may consider the domain of H to be vectors of property values, i.e., $H(p,q) = H(r_1(p), r_2(p), \ldots r_h(p), r_{h+1}(q), \ldots, r_k(q))$.

The *matching score* for a primary p and a parallel q is the value returned by the scoring function H when applied to a primary-parallel pair.

Feature evaluation is the assignment of a value to a feature for a particular primary-parallel pair. This assignment is an expression of the compatibility between p and q with respect to a single feature. For example, if p and q both require the same instrument, then the feature "instruments" would evaluate to 0, indicating that the pair is incompatible with respect to this feature. If different instruments are required, then "instruments" evaluates to 1 for the pair. For certain features, intermediate values are possible. Some features are *qualitative*, i.e., they are either totally present or totally absent. For example, the instruments used by p and q either are or are not the same. The absence of a single qualitative feature may be sufficient to disqualify p and q from simultaneous execution. Other features are *quantitative*, i.e., they are exhibited to varying degrees, possibly on a continuous scale. The feature "timing", for instance, is associated with a goodness-of-fit measure, i.e., the closer the timing requirements of p and q, the better the match.

4. Matching Criteria

Currently, the POMS knowledge based represents the effects of ten features. Five features ("timing", "priority", "roll", "mechanism-motion", "pcs-mode") are quantitative, and the others ("instruments","pointing", "nssc-usage", "permits-parallels", and "manual-match") are qualitative.

"Timing" is the degree to which the primary's time available for parallel science matches the time required by the parallel. The most compatible situation occurs when p has slightly more time available than is required by q. If p is much longer than q, then the match is not as good, since the extra time would be better utilized in a match with a longer parallel. If p is much shorter than the q, then the primary and parallel are clearly incompatible. Total incompatibility does

not exist, however, when p is only moderately shorter than q, since the short-term schedulers may have reason to delay the initiation of the next primary, thereby permitting the parallel to finish. "Timing" is a continuous feature that, in general, assumes values intermediate to the stereotyped situations described above.

"Priority" is number indicating the scientific importance of a parallel SU, as determined by a peer review committee and the Director of the STScI. The higher the number, the higher the compatibility of a parallel with any primary.

When both the primary and parallel have fixed point targets (as opposed to target regions -- see below), it is usually necessary to roll the spacecraft (rotate it about an axis parallel to its bore) in order to bring both targets into the fields of view of both instruments. "Roll" is the number of degrees of spacecraft roll required to do this. Since rolling the spacecraft takes time, the greater the roll, the lesser the compatibility.

The HST has a pointing control system (PCS) that stabilizes the spacecraft during observations. The PCS operates in three different modes, depending on the degree of stability required by the observation. The more stable modes require successively greater overhead time. The policy has been adopted that no matches are permitted between parallels whose PCS requirements are more stringent than the primaries, since the additional overhead would delay the pre-scheduled primary. The best case occurs when the primary and parallel have the same PCS requirement. Matches are permitted in cases where p has a stricter requirement than q, however this is less desirable than the above case, since p's strict requirement is "wasted" on a parallel that doesn't really need it. The feature "pcs-mode" reflects these considerations. If p's PCS mode is greater than q's PCS mode, than p and q are incompatible with respect to "pcs-mode". Otherwise, the greater the difference between the two modes, the lower the compatibility.

The feature "instruments" expresses whether or not the scientific instruments required by p and q are a legal combination for parallel science. Certain instruments are currently precluded entirely, however this is subject to change. If the parallel instrument is allowable and is not the same as the primary instrument, then p and q are totally compatible with respect to this feature.

The feature "pointing" exhibits total compatibility when the targets for p and q are positioned sufficiently close to permit parallel observations. Otherwise, p and q are incompatible with respect to the feature.

Some observations use instruments with moving parts, which causes the HST to vibrate slightly. Other observations cannot tolerate such small vibrations. Obviously, these two types cannot be executed in parallel. The feature "mechanism-motion" exhibits total incompatibility in such cases, and total compatibility otherwise.

An NSSC-1 computer aboard the spacecraft is used to store a variety of temporary data. Since only one instrument at a time may access the computer, p and q are incompatible if they both require its use. The feature "nssc-usage" expresses this criterion.

There are numerous reasons why certain primaries are excluded out of hand from consideration for parallel science. In such cases, the feature "permits-parallels" exhibits total incompatibility.

By setting appropriate fields in the parallel database, the scheduler is permitted to force a match between a particular p and q. In such cases, the highest matching score is assigned to the pair, thereby guaranteeing its delivery to SPSS. In such cases, the feature

"manual-match" exhibits total compatibility.

5. Knowledge representation

The POMS knowledge base contains expertise about the semantics and influences of matching criteria, encoded via a structured set of expert-supplied numerical mappings, or *knowledge functions*. The knowledge functions generate a network, whose traversal implements the application of the knowledge base to a candidate primary-parallel pair. The output of the network is a numerical score that estimates the degree to which the primary and parallel are compatible.

There are four types of knowledge functions, each with a specific representational task:

(1) *measurement functions*, which quantify the degree to which a feature is present in a primary-parallel pair;

(2) *intensity functions*, which normalize the degree of each feature's presence;

(3) *compatibility functions*, which describe relationships between feature intensity and the goodness of a match, with respect to a single feature;

(4) an *aggregation function*, which combines individual feature compatibilities into an overall assessment of the match.

Application of the knowledge base to a primary and a parallel is achieved via the composition of the knowledge functions in a prescribed manner. Figure 2 depicts how this composition may be viewed as a traversal of a network in which the arcs are knowledge functions and the nodes are function inputs/outputs. Processing of the knowledge base corresponds to traversal of the network from bottom to top. The inputs to the network are property values for a primary p and a parallel q. The n measurement functions accept these inputs and return n *measurement values*. The intensity functions accept the measurement values and return n *intensity values*. Compatibility functions accept intensity values and return n *compatibility values*. Finally, the compatibility values are mapped into a single number H(p,q) at the figure's top.

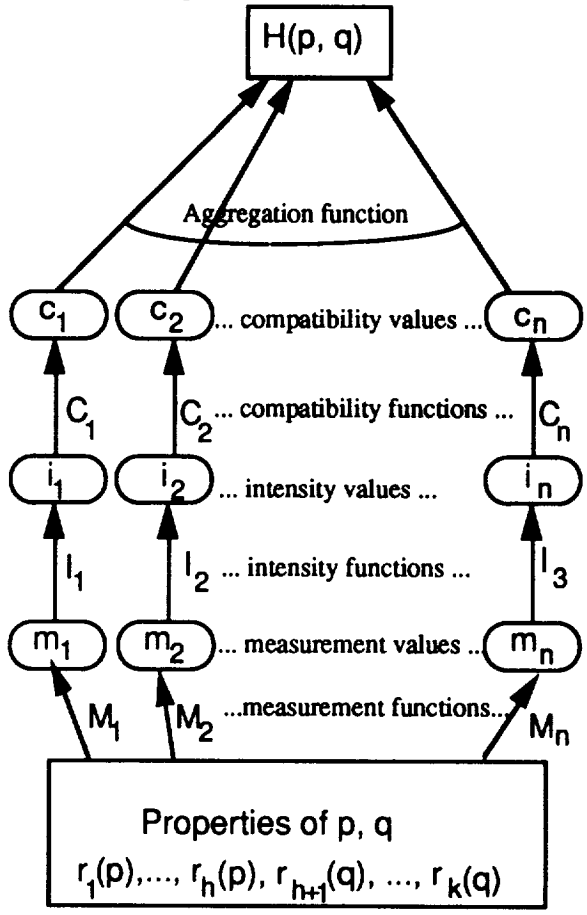


Figure 2. Network model of scoring function H for primary p and parallel q

5.1 Measurement functions

For each feature f_j, there is a measurement function $M_j: P \times Q \rightarrow S_j$ where S_j is the (feature-dependent) range of M_j. M_j is a procedure to measure f_j. The inputs to M_j are property values and the output is a measurement value m_j that expresses (in feature-dependent units) the presence of f_j in a primary-parallel pair. The measurement function for f_j = "instruments", for instance, returns T or NIL, depending on whether the

instruments used by the primary and parallel are different or the same. As an example for a quantitative feature, M_j for the feature f_j = "timing" accepts two inputs: t_q, the length of time (in seconds) required by the parallel for its execution, and t_p, the number of seconds available for parallel science beginning with the start of the primary. t_p and t_q are property values stored in the parallels database. M_j returns a measurement value $m_j = t_p - t_q$, i.e., the difference between the length of time available and the length of time required for a primary-parallel pair.

In certain cases, the measurement function involves more complicated calculations, e.g., the measurement function for the feature "roll" might compute the number of degrees of roll required by the spacecraft to bring both targets into view. The criteria and methods used for measuring features are chosen by the domain expert.

5.2 Intensity functions

For each feature f_j, there is an intensity function $I_j : S_j \rightarrow L$, where S_j is the range of the measurement function M_j. I_j normalizes the measurement value m_j into a uniform scale. In POMS, intensity functions are defined as sets of points provided by the domain expert. For qualitative features, the mapping is frequently trivial, i.e., the measurement value is either 0 (signifying total absence of f_j) or 1 (signifying total presence). For example, if f_j = "instruments" (as described in the previous section), I_j simply maps S_j = {T, NIL} into {0,1} (Table 1). For quantitative features that assume continuous values, the ordered pairs generate a piecewise linear function where additional points may be defined to arbitrarily fine gradation at the discretion of the domain expert. In such cases, a "neutral" intensity value $i_j = 0.5$ indicates moderate presence of f_j. For example, intensity for the feature f_j = "timing", refers to the degree to which the time available exceeds the time required, i.e., if p is a very short primary SU and q is a very long parallel SU, then the feature timing is considered to be weakly present in the pair. Conversely, "timing" is strongly evidenced when a very long primary is matched with a very short parallel, and the feature is neutral when the time available is equal to the time required. This behavior is represented in a table of ordered pairs defining I_j (Table 2). Domain values not explicitly represented in the table are derived by linear interpolation (e.g., $I_j(200) = 0.40$). Since linear interpolation requires finite values, the infinite values at the extremes must be compromised by finite approximations -- in this case 100,000 and -100,000 -- beyond which the function value will not change. The particular data points chosen by the expert are somewhat arbitrary and represent an approximation to the expert's interpretation of the feature's semantics.

Note that intensity is a function only of the feature, not the goodness of a match, i.e., high intensity does not necessarily imply that the primary and parallel are well-matched with respect to the feature. For example, neutral intensity for "timing" (i.e., a situation where the primary and parallel have the same timing requirements) is a more compatible situation than high intensity (i.e., a situation where the time available from the primary greatly exceeds the time required by the parallel). This is illustrated in the following section.

Table 1. Intensity function for feature f_j = "instruments"

m_j	$i_j = I_j(m_j)$
NIL	0.0
T	1.0

Table 2. Intensity function for feature f_j = "timing"

m_j	$i_j = I_j(m_j)$
$-\infty$	0.00
-100,000	0.00
-10,000	0.03
-1,000	0.20
-400	0.30
0	0.50
400	0.70
1,000	0.80
10,000	0.97
100,000	1.00
$+\infty$	1.00

5.3 Compatibility functions

For each feature f_j, there is a compatibility function $C_j : L \rightarrow L$. C_j represents the compatibility between a primary and a parallel with respect only to feature f_j, as a function of the feature's intensity. The input to C_j is the intensity value i_j.The compatibility value $c_j = C_j(i_j)$ represents the goodness of the match between p and q with respect to f_j. Like intensity functions, compatibility functions are represented as sets of ordered pairs. Table 3 shows the definition of C_j for the feature f_j = "instruments". When the feature is absent in the primary-parallel pair (i.e., $i_j = 0$, denoting that the pair uses the same instrument), the pair is incompatible with respect to "instruments". If the feature is present with maximum intensity (i.e., $i_j = 1.0$, denoting that p and q use different instruments), then the pair is judged to be totally compatible ($c_j = 1.0$) with respect to the feature. Table 4 shows a more complicated compatibility function for the quantitative feature "timing".

Table 3. Compatibility function for feature f_j = "instruments"

i_j	$c_j = C_j(i_j)$
0.0	0.0
1.0	1.0

Table 4. Compatibility function for feature f_j = "timing"

i_j	$c_j = C_j(i_j)$
0.0	0.0
0.4	0.3
0.5	0.45
0.55	0.5
0.6	0.7
0.8	0.5
0.9	0.42
1.0	0.25

Here C_j encodes the belief that the optimal compatibility (0.7) is achieved when the intensity is moderately high (0.6). This reflects a situation where the time available is greater, but not too much greater, than the time required. Compatibility declines for higher values because the extra time available would be wasted (and used better with a longer parallel). Situations where the time available is less than the time required are not considered incompatible because SPSS may choose to delay the initiation

of the next primary, in which case the parallel may continue to completion. The compatibility declines sharply as the negative timing disparity increases, however.

The precise values chosen by the expert to define quantitative compatibility functions are not arbitrary. The method for defining these mappings is described in section 7.

5.4 Aggregation function

The knowledge base contains a single aggregation function $A : L^n \rightarrow L$ whose purpose is to combine the compatibility values $c_1, c_2, ..., c_n$ into an overall matching score for a primary-parallel pair. The aggregation function is not a predetermined formula, but rather it is defined by the expert as an explicit part of the knowledge base. It depends on the evaluation features, and may be changed at the expert's discretion. To capture the particular semantics of different features, the aggregation function is built up from operators called *aggregation primitives*, MIN, MAX and α. The primitive operators reflect three different modes of aggregation between features.

For simplicity, we describe each primitive as a binary operator, but since all three are associative and commutative, they may generalized in a straightforward manner to n-ary functions whose arguments may be evaluated in any order. Each primitive accepts two compatibility values c_i and c_j as arguments and returns a value equal to the aggregate effect of c_i and c_j. The semantics of the three operators are shown in Table 5. MIN simply returns the minimum of c_i and c_j, while MAX returns the maximum. MIN and MAX represent cases where the compatibility of either feature always dominates the aggregation, e.g., if $c_i \leq c_j$, then $MIN(c_i, c_j) = c_i$, regardless of the precise value of c_j, i.e., no tradeoffs are exhibited. α is used to express tradeoffs and compensations, i.e., neither feature dominates and the aggregate effect depends on the precise level of each feature.

An example of MIN occurs between the qualitative features f_i = "instruments" and f_j = "permits-parallels". If a candidate primary-parallel pair is incompatible with respect to instruments, (i.e., $c_i = 0$) then the aggregate effect is always incompatible, whether or not the primary permits parallel science. Similarly, if parallel science is not permitted on the primary (i.e., $c_i = 0$), then the aggregate effect is always incompatible, whether or not the instruments are compatible. These semantics are captured by $MIN(c_i, c_j)$. Note that for qualitative features, MIN is equivalent to the logical conjunction (i.e., overall compatibility requires individual compatibility from both features), hence this mode of aggregation is called *conjunctive*.

An example of MAX occurs between f_i = "instruments" and f_j = "manual-match". Since "manual-match" overrides all other features, if a user requests a manual match between a primary-parallel pair (i.e., $c_j = 1$), then the aggregate effect is always totally compatible, regardless of "instruments" compatibility. (Obviously, this feature assumes some special knowledge on the part of the user, and is not to be used carelessly.) These semantics are captured by $MAX(c_i, c_j)$. Note that for qualitative features, MAX is equivalent to the logical disjunction (i.e., compatibility from either feature implies aggregate compatibility), hence this mode of aggregation is called *disjunctive*.

An example of α occurs between the features "timing" and "priority". Regardless of how compatible the candidate pair is with respect to timing, the aggregate effect may be raised (lowered) by the influence of a high (low) priority. The influence of "priority" is similarly modified by the effect of timing compatibility. A significant property of α is that neutral compatibility in either feature has no effect on the aggregation, i.e., $\alpha(c_j, L_N) = \alpha(L_N, c_j)$ for all c_j. This

mode of feature interaction is called *compensatory*.

The formula used to model these semantics depends on the choice of the evaluation interval L and the neutrality point L_N. In POMS (L = [0,1] and L_N = 0.5), we use $\alpha : L \rightarrow L$ defined as

$$\alpha(c_i,c_j) = \frac{c_i c_j}{c_i c_j + (1-c_i)(1-c_j)} .$$

α is a special case of a symmetric sums operator [Sil79]. Its properties are discussed in [Luc90]. This definition for α is not unique, i.e., other formulas might yield qualitatively similar results. See [Che88] for a formal mathematical treatment of related families of aggregation operators. Note that α creates a zero-divide condition when one of the inputs is 0 and the other is 1. To avoid this possibility, compatibility functions that return a value of 1 for any finite measurement value are not allowed for compensatory features. This restriction reflects an assumption that no single compensatory feature is sufficient to dominate the matching process.

The aggregation primitives constitute a simple language for expressing complex interactions among multiple features. In principle, additional operators may be added to the language, although MIN, MAX, and α have thus far sufficed.

Table 5. Behavior of aggregation primitives

c_i	c_j	MAX	MIN	α
high	high	high	high	very high
high	low	high	low	moderate
low	high	high	low	moderate
low	low	low	low	very low

The aggregation function A is defined as a composition of aggregation primitives. This process may be visualized as a parse tree in which the leaves are compatibility values and the internal nodes are aggregation primitives. Figure 3 is an example of an aggregation parse tree for the features f_1 = "manual-match", f_2 = "permits-pars", f_3 = "priority" and f_4 = "timing". Here A = max(c_1, (min (c_2, $\alpha(c_3, c_4)$))). The aggregate effect of "priority" and "timing" is compensatory, and their combined effect is conjunctive with "permits-parallels". Finally, the aggregate effect of the three lower features is disjunctive with "manual-match". For example, if:

(1) a manual match is not requested (i.e., $c_1 = 0$);

(2) the primary permits parallel science (i.e., $c_2 = 1$);

(3) the parallel has been assigned a high priority (e.g., $c_3 = 0.8$);

(4) the timing compatibility is fairly low (e.g., $c_4 = 0.3$);

then the matching score for a candidate pair displaying these features is MAX(0, MIN(1, α(0.8, 0.3))) = 0.63.

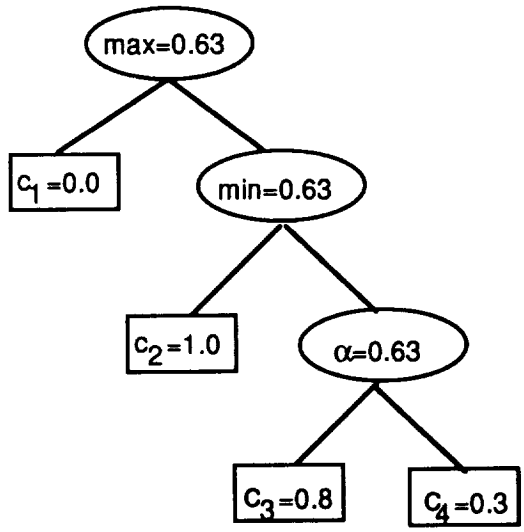


Figure 3. Example of aggregation function parse tree. A(c_1, c_2, c_3, c_4) = 0.63

The full aggregation function (expressed in the n-ary format) currently used in the POMS knowledge base is:

MAX(c_{10}, MIN (c_6, c_7, c_8, c_9, α(c_1, c_2, c_3, c_4, c_5))),

where c_1 = "timing", c_2 = "priority", c_3 = "roll", c_4 = "mechanism-motion", c_5 = "pcs-mode", c_6 = "instruments", c_7 = "pointing", c_8 = "nssc-usage", c_9 = "permits-parallels" and c_{10} = "manual-match". This formula reflects that (1) c1 through c5 are compensatory with respect to one another; (2) c_6 through c_9 and the aggregate of c_1 through c_5 are mutually conjunctive; and (3) c_{10} is disjunctive with all other features.

6. Explanation

Upon request, the matcher generates tables that explain its assessments. Each table summarizes the reasoning that went into the matcher's analysis of the compatibility between a particular primary-parallel pair. An example (reformatted) is shown in Table 6. The header of the table contains the unique SU identification numbers for p and q, and the matching score. In this case, the score of 0.565 indicates that p and q are moderately compatible.

Each row in the table corresponds to the evaluation of a particular feature. The first column contains the feature's name. The second and third columns contain relevant property values for the primary and the parallel, respectively. The primary column in row 1, for example, shows the number of seconds available for parallel science for p (9700), and the parallel column shows the time required by q (700 seconds). For legibility, the name of the property and the units of measurement are not listed in the table, but they are understood by schedulers who are familiar with POMS.

Columns 4, 5 and 6 contain the measurement, intensity and compatibility values, respectively, for each feature. Recalling the "timing" measurement function (Section 5.1), the measurement value in row 1 is the difference between the time available (column 2) and the time required (column 3), or 9000 seconds. By Table 2, this yields an intensity value of 0.95, indicating that there is quite a lot of excess time available. POMS recognizes (via the "timing" compatibility function, Table 4) that this large excess is undesirable and assigns a low "timing" compatibility value (0.33) to the primary-parallel pair.

Table 6. Explanation of match assessment

Primary 0091401 Parallel 0091505 Score = 0.556					
Feature	Primary	Parallel	Mea	Int	Com
timing	9700	700	9000	0.95	0.33
priority	–	3.8	3.8	0.70	0.62
roll	131.4	0, 360	0	0.0	0.50
mech-motion	Y, Y	N, Y	1	0.75	0.53
pcs-mode	fine	fine	0	1.0	0.58
instru-ments	WFPC	FOC	T	1.0	1.0
pointing	2.0, 6.0	r,9.0,5.0 8.0,2.0	T	1.0	1.0
nssc-usage	N	N	N	0.0	1.0
permits-parallels	Y	–	T	1.0	1.0
manual-match	–	–	N	0.0	0.0

For the next feature, "priority", the only significant property is the scientific priority of the parallel, 3.8. No primary properties are relevant, hence the primary column for "priority" is blank. The measurement function for "priority" simply returns the priority property value, hence it is also equal to 3.8. Priorities are assigned on a scale of 1 to 5, hence 3.8 is considered to be a fairly important observation, i.e., the intensity value for "priority" is high (0.70). This high intensity is viewed by POMS as having a

fairly strong positive influence (compatibility value = 0.62).

The primary column for "roll" contains the orientation of the spacecraft (in degrees, relative to an HST-specific coordinate system) that is intended for observing p. In this case, this so=called "nominal orientation" is equal to 131.4 degrees. The parallel column contains an orientation interval that is sufficient to view both the primary and the parallel. In this case, the interval includes all possible orientations, i.e., 0 to 360 degrees. (Any orientation is sufficient because the parallel has a *region target* that contains the primary target. See "pointing", below). The roll measurement function computes the minimum number of degrees that the spacecraft must rotate from the nominal orientation, in order to reach the desired interval. In this case, the nominal orientation is within the desired range already, hence no roll is required. This fact is recorded in the "roll" measurement value (0 degrees). For "roll", this minimal intensity implies neutral compatibility, i.e., the feature will have no effect on the overall assessment (compatibility value = 0.5).

The primary column for "mechanism-motion" refers to two properties: (1) whether or not the primary requires mechanism motion ("Y indicates motion is required); and (2) whether or not the primary can tolerate motion ("Y" means "yes"). The same scheme is used for the q, hence the parallel column for "mechanism-motion" indicates that the parallel can tolerate motion, but does not require it. The measurement function for "mechanism motion" recognizes this situation and returns a tag value of 1 to the measurement value. (The simple tagging scheme for measuring this feature and its related intensity function are not described here.) This combination of motion requirements is considered to be mildly favorable, hence a slight positive compatibility value (0.53) is assigned.

Both p and q require the strictest PCS mode. This is a favorable situation, hence "pcs-mode" exhibits a compatibility value of 0.58.

The remaining features are qualitative. p and q use different instruments (the wide-field planetary camera (WFPC) and the faint object camera (FOC)). Neither uses the NSCC-1 computer, while the primary permits parallel observations, and no manual match has been specified. Hence, the conjunctive features "instruments", "nssc-usage" and "permits-parallels" each exhibit maximal compatibility, and the disjunctive feature "manual-match" exhibits minimal compatibility.

The primary column for the feature "pointing" contains the celestial coordinates (2.0 degrees longitude, 6.0 degrees latitude) of the primary target. The parallel properties indicate that q has a *region target*, i.e., a region of space in which any specific pointing is sufficient. (Region targets are quite common for parallels.) The parallel column describes this region as a rectangular area ("r") centered at 9.0 longitude and 5.0 latitude, with a longitudinal extent 10.0 degrees from center, and a latitudinal extent 2.0 degrees from center. Since the primary pointing is contained well within the parallel's target region, p and q are compatible with respect to pointing. (This also explains why any orientation of the spacecraft is acceptable.)

Application of the POMS aggregation function (Section 5.4) yields a slightly positive matching score of 0.556, despite the poor time-fit. POMS has concluded that the timing problem is outweighed by the combined positive influences a high scientific priority, favorable mechanism motion and pointing control requirements.

7. Knowledge acquisition

In this section, we describe the methods used in POMS for acquiring new expertise and for modifying an existing knowledge base.

7.1 Adding new knowledge

The process of acquiring new knowledge in POMS is relatively structured, compared to conventional knowledge-based systems where acquisition usually requires informal (and often lengthy) dialogs between the domain expert and the knowledge engineer. New knowledge in POMS always comes in the form of a new feature to be added to the assessment process. Support for the new feature requires the expert to provide definitions for the new measurement, intensity and compatibility functions, and to extend the existing aggregation function to include the new feature. To illustrate this process, assume that a new feature f_{n+1} is being added to a knowledge base containing f_1, f_2, ..., f_n.

First, the domain expert selects a name for f_{n+1}, decides on a procedure for measuring the new feature, and determines what SU properties are required for the analysis. The knowledge engineer then implements the procedure in CLOS. A pointer to this code is added to the knowledge base so that it is invoked whenever the new measurement function M_{n+1} is applied.

For the intensity function I_{n+1}, the expert provides a set of ordered pairs as in Section 5.2. For qualitative features, this is straightforward. For quantitative features, the function should reflect the expert's intuitive understanding of the feature's semantics.

Extension of the aggregation function A to incorporate f_{n+1} requires the addition of a new branch and leaf (and possibly a new internal node) to the existing parse tree for A. The expert is requested to identify the modes of interaction between the new and existing features. Based on this analysis, the placement for the new leaf c_{n+1} is identified. (For difficult cases, this ad hoc extension technique may be assisted by a partially mechanical procedure [Luc90].) The engineer then makes a corresponding change in the formula that implements the aggregation function in the knowledge base.

To elicit a new compatibility function C_{n+1} for f_{n+1}, a context is constructed in which the levels of the existing features are constrained such that the new feature's effect completely dominates the aggregate compatibility of $f_1, f_2, ..., f_{n+1}$. Under these assumed constraints, $A(c_1, c_2, ... c_n, c_{n+1})$ is exactly equal to $c_{n+1} = C_{n+1}(i_{n+1})$, for all i_{n+1} (see below). Hence, in the assumed context, $A = C_{n+1}$. To illustrate such a context, we consider the case where all features are compensatory, i.e., where $A(c_1,c_2,...,c_{n+1}) = \alpha(c_1,c_2,...,c_{n+1})$. In this case, the desired context is achieved by assuming that the aggregate effect of $f_1,f_2,...,f_n$ is neutral, i.e., $A(c_1,c_2,...,c_n) = \alpha(c_1,c_2,...,c_{n+1}) = L_N$. Under this assumption, $A(c_1,c_2,...,c_{n+1}) = c_{n+1} = C_{n+1}(i_{n+1})$, for all intensity values i_{n+1}, i.e., the aggregate compatibility of $f_1,f_2,...,f_{n+1}$ is completely dominated by f_{n+1}. This equality, which is easy to verify formally, is consistent with the intuitive interpretation that neutral compatibility has a neutral aggregative effect among compensatory features.

Once the context has been established, the expert is then asked to estimate the matching score for various selected intensity values i_{n+1}, with the other features fixed at their assumed levels. Since $A = C_{n+1}$, each matching score is equivalent to a compatibility value c_{n+1}. Hence, each $<i_{n+1}$, matching score> pair is equivalent to a $<i_{n+1}, c_{n+1}>$ pair. The set of such pairs becomes the new compatibility function C_{n+1}.

The rationale for the above strategy is that it yields the expert's opinion of the direction and degree to which the new feature displaces the neutral effects of the other features. This "strength of displacement" is the essential heuristic used to estimate a feature's significance to the overall aggregation.

Consider a simple case in which f2="priority" is being added to the

existing knowledge base which contains the single feature f1 = "timing". The aggregation function is $A(c_1,c_2) = \alpha(c_1,c_2)$ and the context contains the assumption that $c_1 = 0.5$. The expert is asked to score a match in which timing is neutrally compatible, i.e., $c_1 = 0.5$ for varying intensities of i_2 for "priority". If "priority" is present with minimum intensity (i.e., $i_2 = 0$, the lowest possible priority), then the aggregate matching score is very low, say 0.1. This assessment states the expert's opinion that very low priority significantly degrades the neutral effect of "timing". Some other feature might have a less significant effect. At neutral intensity for "priority" (i.e., $i_2 = 0.5$), the matching score might be 0.5, implying that a moderate priority has neither a positive nor negative effect on the matching score. The highest priority (i.e., $i_2 = 1.0$) might create an aggregate score of 0.8, representing the opinion that high priority has a strong positive effect on compatibility. The data pairs [(0, 0.1), (0.5, 0.5), (1.0, 0.8)] are included in the new compatibility function C_2. Additional points may be provided to whatever granularity is deemed necessary by the domain expert.

The assumed context of aggregate neutrality described above is appropriate only if all features are compensatory. Different assumptions are required for non-compensatory features. If f_1, $f2,...,f_n$ is conjunctive with respect to f_{n+1}, then the aggregate effect of $f_1,f_2,...,f_n$ is assumed to be totally compatible (i.e., $A(c_1,c_2,...,c_n) = 1.0$). In this case, $A(c_1,c_2,...,c_{n+1}) = MIN(A(c_1,c_2,...,c_n), c_{n+1}) = MIN(1.0, c_{n+1}) = c_{n+1}$. Hence, the aggregate effect of $f_1,f_2,...,f_{n+1}$ is dominated by f_{n+1}, as desired. In this case, the matching scores provided by the expert represent the new feature's strength to displace the total compatibility of $f_1,...,f_n$.

Disjunctive features are handled by assuming that $f_1,...f_n$ are present with minimal compatibility, and the matching score represents the new feature's strength in displacing the minimal compatibility of $f_1,...,f_n$.

In the general case, where all three modes of interaction may be present in $f_1,...f_n$, the context requires all three forms of assumptions. For example, suppose that the feature f_4 = "timing" is added to an existing knowledge base containing f_1 = "manual-match", f_2 = "permits-parallels" and f_3 ="priority". The modes of interaction among these features is shown in Figure 4. The context for defining C_4 consists of the following assumptions:

(1) f_3 exhibits neutral compatibility (the parallel has moderate priority, i.e., $c_3 = 0.5$);

(2) f_2 exhibits total compatibility (the primary permits parallel science, i.e., $c_2 = 1$);

(3) f1 exhibits total incompatibility (a manual match is not requested, i.e., $c_1 = 0$).

It is easily verified that under these assumptions $A(c_1,c_2,c_3,c_4) = C_4(i_4)$, hence by describing the overall compatibility with variances in the intensity of f_4, the expert expresses a compatibility function for f_4.

The method described above is a heuristic for approximating compatibility functions. The degree to which the defined function actually represents the true effects of a feature depends on how closely the feature's general behavior is modeled by its behavior under the assumed constraints. In certain cases, more restrictive assumptions (such as assigning specific measurement values to certain features) are necessary in order for the context to make sense to the domain expert [Luc90]. Other heuristics are possible, but in practice the above strategy has worked satisfactorily.

7.2 Changing existing knowledge

Any of the existing knowledge functions may be modified incrementally. Most

commonly, changes are made to compatibility functions in order to fine-tune the relative effects of features. Suppose, for example, that the scheduler, upon review of many matches, decides that POMS is underscoring matches in which p has a large excess of time available for parallel science (as in the example of Section 6). This deficiency is easily addressed by dampening the negative values defined for such cases in the "timing" compatibility function. For instance, the last entry in the function definition (Table 4) might be adjusted from (1.0, 0.25) to (1.0, 0.30), thereby lessening the worst case effect of the feature. The second-from-last entry might also be modified slightly from (0.9, 0.42) to (0.9, 0.45). Using this modified compatibility function for "timing", the example in Section 6 would yield a matching score of 0.602, somewhat higher than the original score of 0.556. Changes to intensity functions (reflecting a reassessment of a feature's semantics) are made in a similar manner.

Currently, such changes are made by manually editing a file that contains the knowledge base, although an interface is planned that will permit such changes to be made interactively by the scheduler.

Changes in the way features are believed to interact are made by rewriting the aggregation formula in the knowledge base file. Changes to a feature's measurement function usually involves the recoding of the CLOS procedure that implements it. This requires intervention by Spike system developers.

Features may be deleted from the knowledge base simply by removal from the argument list of the aggregation function.

8. Results

Since the HST parallel science program is still in the early testing stage, it is too early to make conclusive statements about the performance of POMS. Currently the system has been used successfully to verify matches made by human schedulers. POMS has also been employed to analyze the frequency of good matches in a large pool of proposals to be executed in 1992. Although much improvement is required before POMS can assume a more autonomous role in scheduling, the system's rate of improvement has been highly encouraging thus far.

The most apparent strength of the system in this early phase (during which it has been subjected to frequent changes in the matching criteria) has been its capability for incremental refinement. The explanation facility has been quite useful in identifying assessment errors and these errors have, in most cases, been easily corrected by adjusting a compatibility function. Furthermore, the corrections have, in almost all cases, been made without destroying the prior integrity of the knowledge base, i.e., without invalidating previously correct assessments. This same amenability to local refinement was also observed in the previous application of the technique [9,10] and seems to be a generic advantage of the approach. As the human knowledge sources for POMS become more familiar with the parallel science problem, we expect that POMS will be able to represent and use this expertise in an accurate fashion.

There is no particular dependence between the POMS methodology and the parallel science problem. Hence, if successful in the present arena, the knowledge representation scheme should be applicable to other problems involving the assessment of multiple quantitative criteria. We are currently investigating several potential applications, including the detection of duplicate scientific requests in the HST proposal pool, and the matching of scientific observations to point spread functions for image restoration.

Acknowledgements

The parallel science evaluation criteria were originally identified by Mark Johnston of the STScI Science Engineering and Scheduling Division. Jim Mainard, John Baum, John Isaacs and Brian Ross were instrumental in developing the POMS knowledge base. The author is particularly grateful to Glenn Miller, Jeffrey Sponsler, Tony Krueger and Mark Giuliano for many useful suggestions and for their careful readings of the draft.

References

Cheng Y. and Kashyap R. L. (1988). An Axiomatic Approach for Combining Evidence from a Variety of Sources. *Journal of Intelligent and Robotic Systems*, 1, 17-33. [Che88]

Gerb A. (1991). Transformation Reborn: A New Generation Expert System for Planning HST Operations, *Telematics and Informatics*, to appear. [Ger91].

Johnston, M., Miller, G., Sponsler, J., Vick, S., and Jackson, R. (1990). Spike: Artificial Intelligence Scheduling for Hubble Space Telescope". In S.L. O'Dell (ed.). *Proceedings of the Fifth Conference on Artificial Intelligence for Space Applications* (pp. 11-18). NASA Conference Publication 3073. [Joh90a].

Johnston, M. (1990). SPIKE: AI Scheduling for NASA's Hubble Space Telescope. In *Proceedings of the Sixth IEEE Conference on Artificial Intelligence Applications* (pp. 184-190). Los Alamitos, California: IEEE Computer Society Press. [Joh90b]

Kinney, A. L. and Blades, J. C. (Eds.). (1991).*The First Year of HST Observations*. Space Telescope Science Institute, Baltimore, Maryland. [Kin91].

Lucks, M. and Gladwell, I. (1992). "Automated Selection of Mathematical Software, *ACM Transactions on Mathematical Software*, 18 to appear. [Luc92].

Lucks, M. (1990). A Knowledge-Based Framework for the Selection of Mathematical Software, Ph. D. dissertation, Dept. of Computer Science, Southern Methodist University. [Luc90].

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. *Telematics and Informatics* , 5, 197-212. [Mil88].

Miller, G. and Johnston, M. (1991). Long Range Science Scheduling for the Hubble Space Telescope, *Telematics and Informatics*, to appear. [Mil91]

Silvert, W. (1979). Symmetric Summation: A Class of Operations on Fuzzy Sets. *IEEE Transactions on Systems, Man and Cybernetics* 9(10), 657-659. [Sil79]

Taylor, D. K., Reinhard, K. E., Lanning, H. H. and Chance, D. R. (1991). The Scheduling of Science Activities for the Hubble Space Telescope. In A. L. Kinney, J. C. Blades (Eds.). *The First Year of HST Observations* (pp. 281-287). Space Telescope Science Institute, Baltimore, Maryland. [Tay91]

Monitoring/Control/Diagnosis

Coordinating Complex Problem-Solving Among Distributed Intelligent Agents

N92-23360

Richard M. Adler
Symbiotics, Inc.
875 Main Street
Cambridge, MA 02139

P. 11

ABSTRACT

This paper describes a process-oriented control model for distributed problem-solving. The model coordinates the transfer and manipulation of information across independent networked applications, both intelligent and conventional. The model was implemented using SOCIAL, a set of object-oriented tools for distributed computing. Complex sequences of distributed tasks are specified in terms of high-level scripts. Scripts are executed by SOCIAL objects called Manager Agents, which realize an intelligent coordination model that routes individual tasks to suitable server applications across the network. These tools are illustrated in a prototype distributed system for decision support of ground operations for NASA's Space Shuttle fleet.

Keywords: distributed control, intelligent coordination, distributed artificial intelligence

INTRODUCTION

End-user tasks in distributed systems typically decompose into sequences of interactions between independent applications. For example, scheduling engines are often driven by task, resource, and constraint networks derived from independent planning systems. Scheduling a space mission may therefore depend on a succession of individual data transfers and manipulations across several decision support tools and databases. Similar task decompositions arise in operations support for complex control networks such as the Space Shuttle Launch Processing System (Adler, 1990).

Interactions among distributed applications and data stores must be initiated and managed. In the absence of direct interprocess links, human intervention is required to effect transfers of data and control. Such involvement, whether by end-users or supporting network operators, impacts the productivity and cost of frequent, high-level activities such as decision support. Moreover, the likelihood of human errors may compromise quality and safety.

Intelligent systems have the capacity to coordinate distributed problem-solving autonomously. However, considerable latitude exists in designing architectures for distributed intelligent control (Bond and Gasser, 1988). For example, interaction sequences can be automated piecemeal, by establishing directed, data-driven control links between individual applications. Distributing sequential control logic in this manner is cumbersome in application networks that address multiple complex tasks. Moreover, directed links are difficult to maintain, extend, and verify when network applications and tasks are added or modified with any frequency. Finally, highly distributed control schemes incur processing overhead to ensure focus and coherence of autonomous problem-solving activities.

This paper describes a process-oriented model for distributed coordination. The model enables complex sequences of distributed tasks to be specified in terms of high-level scripts. Each script element represents a distinct data transfer task or request for problem-solving skills between complementary applications. The model also encompasses intelligent control modules that execute these process scripts automatically: individual tasks are routed to suitable distributed servers and results

are retrieved for the requesting applications. This model alleviates many of the difficulties faced by more decentralized coordination schemes.

The new process control model was implemented as an extension to SOCIAL, a development tool for distributed computing across heterogeneous hardware and software environments. The next section of the paper reviews SOCIAL's architecture and functionality. The following section describes the design and implementation of the process control model. The model is then illustrated with a prototype distributed system for decision support of Space Shuttle fleet ground operations at the NASA Kennedy Space Center.

OVERVIEW OF SOCIAL

SOCIAL consists of a layered collection of object-oriented tools for distributed communication, data management, and control (cf. Figure 1). These generic capabilities are bundled into active objects called *Agents*. SOCIAL provides an extensible library of predefined Agent classes with specialized integration and coordination behaviors. An application is linked non-intrusively to an Agent via calls to a high-level Application Programming Interface (API). Applications make API calls to invoke their mediating Agent objects to execute desired distributed behaviors. Agents interact using asynchronous message-passing. SOCIAL's underlying layers transparently manage interprocess message communication across heterogeneous languages, operating systems, and networked hardware platforms (Symbiotics, 1990).

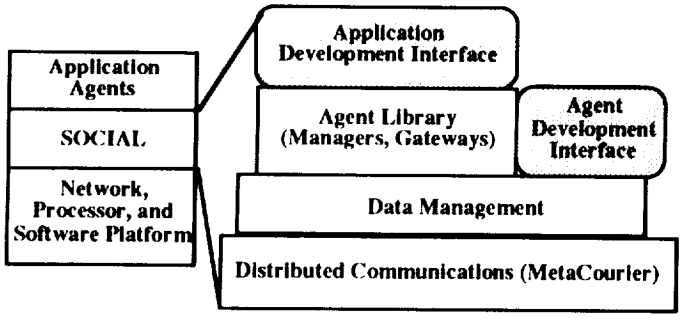


Figure 1: Architecture of SOCIAL

SOCIAL Gateway Agents

SOCIAL Gateway Agents provide a uniform design model and methodology for integrating heterogeneous applications, both conventional and intelligent (Adler, 1991b). The root Gateway Agent class defines a full peer-to-peer control model that is inherited by all specialized Gateway subclasses. This model invokes a set of Agent methods in a data-driven manner to process: (a) outgoing messages from the Gateway's associated application to other Agents; (b) incoming messages from other Agents; and (c) responses to prior outgoing messages.

An application is integrated by creating a new Gateway subclass, which involves specializing two sets of Agent methods. One set establishes custom mappings for application-specific data models and control interfaces. To simplify interactions between heterogeneous applications, SOCIAL transports information in a neutral exchange format. Accordingly, each Gateway subclass must define conversion methods for translating from the application's native knowledge representation model and command interface into SOCIAL's neutral data format, and vice versa. Native and neutral exchange data structures are accessed and manipulated using functions from the application's programmatic interface and the API for SOCIAL's data management layer.

The second set of Gateway methods defines the application's desired interactions with other elements of the distributed system. These control methods are constructed using the Gateway conversion methods for extracting data and knowledge, injecting information, or invoking application commands, as required. One method establishes ***server*** behaviors, which process incoming messages from other application Gateways and generate suitable responses. A second method defines ***client*** behaviors. Applications configured as clients initiate outgoing messages containing service requests via their Gateways. A Gateway client behavior typically injects responses to previous request messages back into the associated application for follow-on processing. A given application Gateway can support multiple client and server interactions with any number of other application Agents.

SOCIAL Manager Agents

SOCIAL Manager Agents provide predefined control models for coordinating activities in complex networks of application Agents. Coordination among distributed problem-solvers can be achieved through different strategies. One approach is to distribute control, localizing it within individual applications. A second approach is to centralize control, either in a preferred application or in a dedicated, independent module. SOCIAL Gateway Agents provide flexible vehicles for implementing either of these opposing alternatives. Manager Agents were developed to support a third design strategy, which is to combine localized and centralized control into ***hybrid*** coordination architectures.

The first SOCIAL Manager Agent defined a hierarchical, distributed control (HDC) model (Adler, 1991a). This HDC-Manager mediates interactions among autonomous "subordinate" Agents much like a human manager. Application Gateways communicate exclusively with their Manager Agent, requesting information or problem-solving resources, and receiving responses to those requests. Subordinate Agents do not need to know about the functionality, structure, or even the existence of other application Agents; they only need to know (a) the high-level API for interacting with the HDC-Manager and (b) the names of the services available within the HDC-Manager's scope.

The basic operational model for the HDC-Manager is summarized in Figure 2. The HDC-Manager functions as an intelligent router of task requests, based on a directory knowledge base. This directory describes: (a) individual information resources and problem-solving capabilities; (b) the application Agent that supports each such service; (c) the message format for requesting that service; and (d) the server Agent's logical address. Request messages from application Gateway Agents are posted to the HDC-Manager's agenda queue. The HDC-Manager processes and dispatches requests asynchronously to suitable server application Gateways. These Agents, in turn, post responses from their applications to the HDC-Manager's "bulletin-board" database. The HDC-Manager subsequently retrieves such responses and forwards them back to the original requesting Agents.

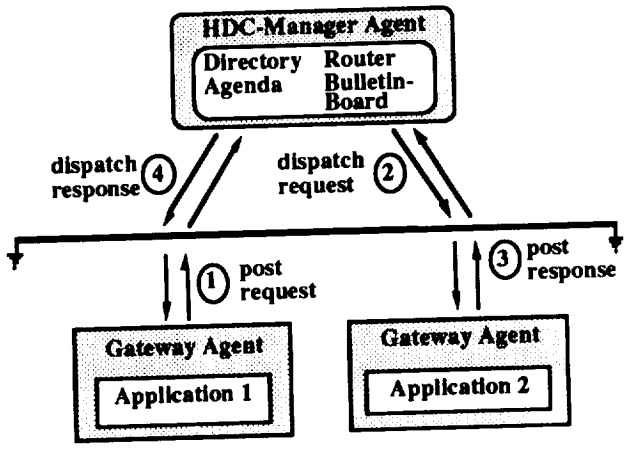


Figure 2: HDC-Manager Operational Model

In essence, the HDC-Manager establishes a layer of control abstraction that decouples application Gateways from direct connections with one another. The centralized directory promotes maintainability and extensibility over the evolutionary lifecycle of complex distributed systems.

SOCIAL'S PROCESS-PLANNER AGENT

SOCIAL's HDC-Manager Agent supports simple interactions between independent distributed systems. For example, an intelligent scheduling tool might query a remote shop floor production database to determine the availability of equipment or labor resources. Similarly, an intelligent operations support application for a power management system might collect data to confirm a power bus fault hypothesis, or command an experiment management system to minimize power consumption.

The applications in these examples are loosely coupled. The scheduler uses the database solely as a source of current status information about its target domain. Similarly, operation management systems only interact in situations where the structural and functional interfaces between their target subsystems appear to be relevant. Simple ***discrete*** transactions (e.g., query-response, sensor polling, command-acknowledgment interchanges), provide

sufficient coupling to enable distributed problem-solving activities in these contexts. The HDC-Manager Agent contains all of the apparatus and control functions required to coordinate discrete transactions within a distributed system.

However, many kinds of distributed problem-solving activities cannot be accomplished within the scope of an individual logical transaction between two remote applications. Consider, for example, a distributed decision support system composed of two or more independent tools, such as a planning system and a scheduling engine. Suppose that the planning system incorporates the master database for all decision support information, including all operational plans and schedules for the target domain. Assume also that the models used to represent data and knowledge are incompatible across the two tools, which is common for independent systems specialized to solve different problems.

In this context, an elaborate set of information and control exchanges has to take place to perform scheduling. Data must be extracted from the planning system's database, transferred to the scheduler, translated into a format that is compatible with the scheduler, and then loaded. At this point, the primary scheduling activity itself can proceed. Once scheduling has been completed, a similar set of support transactions must be accomplished in reverse order. The completed schedule must be translated into a format acceptable to the planner, transferred back to the planner's host platform, and incorporated into the master decision support database.

Clearly, such sequences need to be automated, both for end-users and for autonomous management systems. It should be possible to invoke scheduling or comparable functions through simple, high-level commands. Such commands should specify only the few data items that are required to characterize particular instances of the desired task type (e.g., a mission identifier, options to override default control parameters for the scheduling engine). This amounts to a requirement to automate ***composite*** activities, or ***processes***, composed of multiple discrete interactions between independent distributed applications. The HDC-Manager Agent currently lacks the requisite capabilities either to define such distributed problem-solving processes or to coordinate their execution. We considered several design approaches to extend SOCIAL to provide the desired functionality.

One alternative would be to configure a distributed system so that one message would initiate the desired activity sequence by triggering the first application Gateway to perform its assigned task, pass the results onto a second Gateway to perform the second required task, and so forth. In other words, a single message to the first server Agent would automatically initiate the desired chain of interactions. SOCIAL's communication layer maintains a "travel log" for each message as it is passed through Agents. Once the "terminal" Agent completes its activities, results are automatically returned and post-processed through all preceding Agents that appear in the message's log.

Unfortunately, the logic for parsing and forwarding messages within individual application Gateway Agents can become quite involved for complex processes. Moreover, a given application Agent may have to perform a given function within multiple process sequences, with different successor Agents and post-processing activities for each distinct chain. Maintainability and extensibility are compromised in that each time a new process is defined, the control logic for Gateway Agents in that process chain must be modified. Consequently, for mission critical applications, the entire suite of behaviors for each affected Agent has to be verified again. Finally, SOCIAL's communication model only supports ***acyclic*** message forwarding, precluding processes involving "back-and-forth" exchanges or iterative looping.

A second alternative would be to extend the HDC-Manager directly to support the specification and execution of distributed sequential processes. On this strategy, special "macro" tasks would be definable in the HDC-Manager's directory knowledge base, corresponding to composite processes. A message requesting execution of such a composite task would activate extended control logic.

This logic would decompose macro tasks into constituent service requests and post individual steps to the task agenda, in suitable order, for the HDC-Manager to process and route.

This approach resolves the objections raised against the previous design strategy. First, messages are only passed between the extended HDC-Manager and individual application Agents, eliminating the hardwiring of interaction sequences directly into the control logic for individual Gateways. Second, the new macro processes are modular, maintainable, and readily extensible. In particular, processes are modeled independent from and external to individual application Agents. System testing is simplified because new processes can now be defined without affecting previously verified processes and Agent behaviors. Finally, the extended HDC-Manager mediates all interactions between application Gateways as separate message transactions. SOCIAL's acyclic message-passing model can accommodate cyclic behaviors that are broken up in this manner.

The main objections to extending the HDC-Manager are performance and complexity. This Agent's primary design role is to eliminate direct connections between application Agents by mediating interactions. The proposed functional extensions decompose macro tasks and manage queueing of process subtasks. These capabilities for managing distributed processes impose computational overheads that reduce the responsiveness of this core routing capability. Moreover, these design extensions also complicate the original control logic of the HDC-Manager significantly.

We adopted a third approach, which distributes the functionality of the extended HDC-Manager to overcome these design problems. Specifically, centralized process definition and management functions are retained, but ***decoupled*** from the HDC-Manager and assigned to a new subclass of Manager Agents called Process-Planners. The distributed control model realized in the Process-Planner is then configured to ***drive*** the HDC-Manager through the individual process steps comprising composite activity sequences. It does this by posting successive service requests to the HDC-Manager's agenda for distributed routing. This basic architectural configuration is depicted in Figure 3.

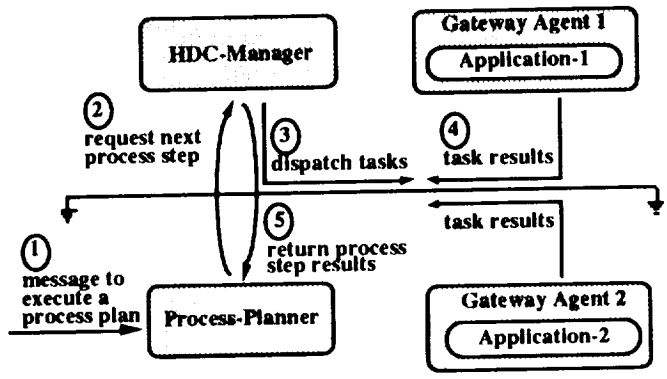


Figure 3: SOCIAL Process Control Architecture

This distributed design is attractive because it enables the HDC-Manager to function identically for two distinct distributed computing models – transaction-based and process-based. The Process-Planner manages process decomposition and activity sequencing. It transmits individual process steps as individual service requests to the HDC-Manager, replicating the type of inputs that would be expected from ordinary application Gateway Agents. Consequently, the HDC-Manager need not distinguish between discrete and composite service requests within its agenda. In fact, process steps and service requests representing discrete Gateway transactions can be interleaved on the HDC-Manager's agenda, enabling both kinds of interactions to be coordinated concurrently. In addition, partitioning distributed control logic across two Manager Agents fosters modularity, maintainability, and extensibility.

The message traffic between the Process Planner and HDC-Manager entails some performance overhead. However, the two Agents communicate asynchronously and can operate concurrently on dedicated processors, compensating at least in part for message-passing overhead. Overall performance depends strongly on the particular distributed system and its ratio of communication and coordination to local application problem-solving loads.

Implementing the Process-Planner Agent

The Process-Planner Agent was implemented as a subclass of SOCIAL Gateways. Consequently, it inherits the standard Gateway peer-to-peer control model and API methods. These methods were specialized to interface with the core process planning application. The data injection API method parses two SOCIAL neutral exchange types. Character strings are interpreted as pathnames for files containing process scripts, which the planning system loads into memory. Lists are treated as commands and command arguments, which are executed through the application's control interface (e.g., initialize, reset). Extensions to handle other data types amount to straightforward program Case statement clauses. The API method for extracting information was not required, because the process planning system functions solely in a client role.

The process planning application examines a process script to determine the next step to perform. Currently, a script consists of an ordered list of entries that correspond to services identified in the directory knowledge base of the associated HDC-Manager Agent. The :compute-next-step command retrieves the first script item that has not been instantiated. An item has been instantiated if it has been annotated with execution results returned from the HDC-Manager.

The planning program also computes predecessors and successors to current steps in process scripts. The HDC-Manager supports a generic file transfer service based on SOCIAL utility agents that send and receive files across network nodes. This service is context-sensitive in that it presupposes source and target file pathnames and host names. The HDC-Manager can determine these items given the previous and succeeding script steps to the current file transfer task.

The Process-Planner Agent starts up the embedded planning program in response to a message that specifies initialize and reset commands, together with the name of a script file to load. A second message initiates the following control cycle:

1. the Agent determines the next process step from the process planner program and dispatches a suitable service request message to the HDC-Manager;
2. the HDC-Manager dequeues the request from its agenda, finds the server application Agent (e.g., a Gateway for a scheduling engine, a Send-File utility), and dispatches an appropriate task message to that Agent;
3. the target application Agent performs the assigned task and posts its response to the HDC-Manager's bulletin-board. Typically, the target Agent is a Gateway, which interacts with its embedded application by injecting data or commands and collecting query or problem-solving results;
4. the HDC-Manager automatically routes the posted task results back to the Process-Planner Agent;
5. the planning program updates the instantiated script with service request results and computes the next step from the script. The Process-Planner dispatches this new process step back to the HDC-Manager for routing.

The Process-Planner then reiterates this execution cycle. The Agent terminates looping when notified by its embedded process planning program that the script has been completely instantiated.

The Process-Planner plays the role of a Manager Agent in that it performs distributed control functions rather than integrating domain-specific applications. However, it was designed and implemented as a subclass of Gateway Agents. Sophisticated process planning tools are beginning to appear commercially in CAE, CAM, and CASE domains. These tools are used to specify task decompositions and automate control of work flows for machining complex parts, other manufacturing processes or managing large projects. The Gateway's uniform, high-level interface architecture preserves design flexibility to replace the SOCIAL process planning program with a more powerful dedicated engine.

Distributed Decision Support Prototype

A prototype was developed to validate this design model for coordinating processes in SOCIAL. This prototype simulates a distributed decision support system for ground operations activities for the Space Shuttle fleet at the NASA Kennedy Space Center. Specifically, a Process-Planner Agent was implemented and coupled to an HDC-Manager. These two Agents automatically coordinate the complex sequence of distributed activities required to schedule Shuttle missions.

Two (simulated) decision support applications were integrated using SOCIAL Gateway Agents (cf. Figure 4). One Agent represents a commercial planning system called Artemis, which NASA has modified with a frontend interface customized for planning ground support activities for Shuttle missions. The second Agent represents an intelligent, constraint-based scheduling engine called Gerry (Zweben, 1990), which is being developed by the NASA Ames Research Center. Artemis is based on a proprietary fourth generation language and resides on an IBM mainframe host. Gerry, written in Common Lisp and CLOS, runs on Unix workstations.

The Artemis Gateway Agent is configured to simulate three tasks: (a) downloading data files for a particular mission from the Artemis master planning database; (b) uploading data files representing a completed mission processing schedule back into Artemis; and (c) running an analysis program to detect and report resource conflicts between the new schedule and existing schedules for other Shuttle missions. The Gerry Gateway also simulates three tasks: (a) translating and loading mission plan files into the scheduler; (b) computing the mission schedule; and (c) extracting and translating the completed schedule back into Artemis-compatible file format.

The Gerry scheduler requires four types of plan information: a network of tasks to be performed to prepare the Shuttle vehicle and its associated payload(s) for launch; a specification of available resources (e.g., labor schedules, equipment such as cranes, and other materials); a set of constraints on tasks and resources; and a data dictionary that describes the information fields in the preceding three datasets. Artemis generates these datasets as four ASCII files in a standardized record format. Gerry requires data to be input from ASCII files in a custom object-oriented format.

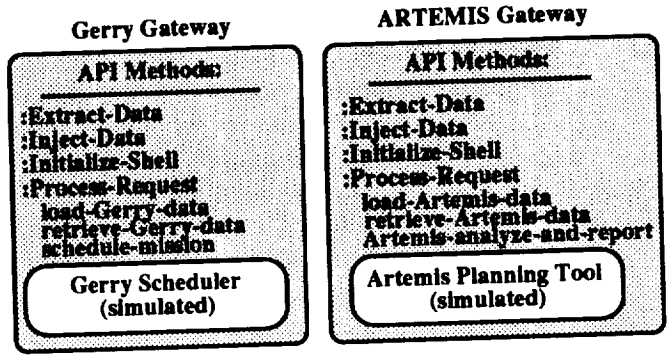


Figure 4: Prototype Decision Support Gateways

SOCIAL's data management subsystem was used to define custom neutral exchange data structures. Translators were written to map between Artemis and SOCIAL data models and between Gerry and SOCIAL data models (cf. Figure 5). The translators were hooked into the application Gateway Agent interface API methods to perform appropriate conversions of data file formats. Data files are translated into neutral exchange format structures in memory, and then written to new files in the target converted format.

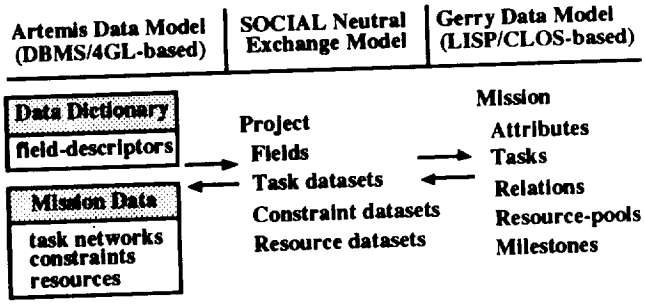


Figure 5: Disparate Decision Support Data Models

A subclass of HDC-Manager Agent, called the DSS-MGR, was created to coordinate interactions

between the two decision support applications (cf. Figure 6). Three steps were required to specialize the DSS-MGR Agent for this purpose. First, conditions were defined for prioritizing agenda service requests. The DSS-MGR sorts requests with respect to an ordinal list of service types. Requests of the same type are ordered by increasing values of a numeric priority attribute. Second, the DSS-MGR directory knowledge base was constructed. The directory identifies all services available from all application Gateways subordinate to the DSS-MGR, plus the generic file transfer capability. Both DSS-MGR attributes are defined using the high-level declarative API specific to HDC-Manager Agents. Third, dispatching functions were written for each directory service entry. These functions manipulate data arguments contained in service request messages into a task message that the HDC-Manager routes to the relevant application Gateway server.

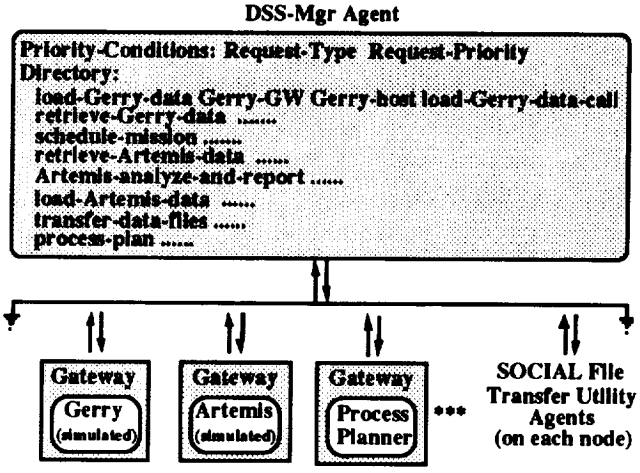


Figure 6: Decision Support HDC-Manager Agent

Next, a script was written for the Process-Planner Agent, defining the distributed process for scheduling Shuttle missions. This sequence consists of the following steps:

```
(retrieve-Artemis-data)
(transfer-data-files)
(load-Gerry-data)
(schedule-mission)
(retrieve-Gerry-data)
(transfer-data-files)
(load-Artemis-data)
(Artemis-analyze-and-report)
```

File format conversions currently take place within the load-Gerry-data and retrieve-Gerry-data tasks. Once the various Agents are loaded and initialized, the mission scheduling sequence is initiated through a simple message to the Process-Planner Agent to compute the next step for a particular mission, such as STS-40. The Process-Planner Agent then executes the control loop described in the previous section against the mission scheduling script.

All demonstration Agents and simulated applications were written in Common Lisp. The demonstration system can be run on a single platform or combination of platforms that currently run SOCIAL/Lisp, including Apple Macintosh IIs, Lisp Machines, and Unix workstations. The Agent Library is currently being converted to C, to run on SOCIAL/C workstation hosts. A planned port of SOCIAL/C to the IBM/VM environment will establish direct interprocess interfaces across mainframes and workstations. NASA's distributed decision support system can then be implemented on the intended target platforms.

FUTURE DIRECTIONS

The Process-Planner Agent is being redesigned with extended functionality. The original planning program only supports simple sequential scripts. These scripts cannot specify data-driven processes, in which successive steps are determined dynamically at runtime, contingent on the results of preceding process steps. Moreover, the initial Process-Planner drives the HDC-Manager to execute script steps individually, in a strictly synchronous, "execute and wait" sequence. Ideally, the Process-Planner should be able to request the HDC-Manager to route all script activities that are mutually independent within a single control cycle. To overcome these limitations, a more expressive scripting language will be developed for specifying processes that incorporate conditional branching, iteration, and concurrent tasking. The Process-

Planner's control logic will be extended accordingly. A capability for executing multiple scripts simultaneously will also be added.

A second set of enhancements will provide more formal development tools for creating and managing script libraries, replacing the ad hoc techniques used in the prototype Process-Planner. A menu-based editor will be developed to access and manipulate process scripts. Also, scripts will be stored in a central database of process plans rather than in an arbitrary collection of independent files.

Other development efforts will extend SOCIAL's library of Manager Agents. The current HDC-Manager is adequate for distributed systems in which a single application Agent represents the unique source for a given resource or service. However, additional control requirements arise for application networks in which ***multiple*** application Agents can provide data, knowledge, or problem-solving skills redundantly. For example, identical copies of a program may be available on several nodes. In addition, some applications may have overlapping functionality for planning, scheduling, or other tasks. A dedicated "Server-Group" Agent, inspired by the ISIS model for group-based tasking (Birman, 1990), is being designed to address distributed control issues for functionally redundant application networks. Like the Process-Planner, this Agent will be configured to offload new control capabilities and work cooperatively with the HDC-Manager. Specifically, the Server-Group will monitor availability of server Agents, determine the best server for a task, and enable redundancy-based approaches to fault tolerance.

RELATED WORK

Alternative frameworks for developing heterogeneous, distributed intelligent systems include ABE (Hayes-Roth, 1988), MACE (Gasser, 1987), Agora (Bisiani, 1987), and Cronus (Schantz, 1986). MACE incorporates dedicated manager agents for centralized routing of messages among application agents. However, MACE managers lack the other capabilities of SOCIAL HDC-Managers, such as shared memory, transparent returning of message responses, and extensibility for multi-level control hierarchies. Like SOCIAL, ABE, Agora, and Cronus all provide virtual environments to shield users from platform dependencies and networking mechanics. However, they do not implement generic distributed services in uniform, object-oriented layers that are accessible to developers for customizing. Agora relies on communication through shared-memory, reflecting its orientation towards parallel multi-processing architectures. The other tools use message-passing models comparable to SOCIAL. ABE and Agora provide predefined control frameworks such as data flow and blackboard models. Unlike SOCIAL Manager Agents, these models explicitly couple individual applications directly to one another. Moreover, heterogeneous SOCIAL Manager Agents can be configured to work together cooperatively. It is unclear whether the other tools support such combinations within a given distributed systems.

The literature on distributed artificial intelligence (DAI) contains many interesting architectures for cooperative problem-solving, including blackboard systems, contract nets, and collections of autonomous agents (Bond and Gasser, 1988). In this context, cooperation refers to loosely-coupled networks of intelligent Agents working to solve a single complex problem through collective action. Most such DAI architectures rely on purely localized control models duplicated across homogeneous, autonomous agents. These designs can be replicated within the generic communication and control model provided by SOCIAL Gateway Agents.

More recent DAI research focuses on theories of cooperation for open-ended systems composed of arbitrarily heterogeneous applications (Gasser and Huhns, 1989). The critical problem here is to design dynamic interaction protocols for communicating self-descriptive goals, plans, and intentions among agents with radically different knowledge and perspectives. SOCIAL Managers currently address more modest closed-world domains, in which the resources available in an agent network are specified ***a priori*** and statically. A synthesis of Manager control models with dynamic interaction pro-

tocols could contribute to a powerful theory of cooperation for open networks of autonomous agents: agents and their resources could be registered dynamically in the context of a partially centralized control architecture that mediates agent interactions.

CONCLUSIONS

SOCIAL applies a highly modular, ***non-intrusive*** object-oriented approach to simplify the design and implementation of complex distributed systems (cf. Figure 7). High-level Agent APIs partition generic distributed computing and application-specific functionality. Gateway Agents provide a uniform methodology and design architecture for integrating heterogeneous applications, both intelligent and conventional. Manager Agents provide high-level distributed control building blocks for tying application Gateways together. Coordinating via Managers eliminates direct connections between individual application Agents that are difficult to maintain and extend.

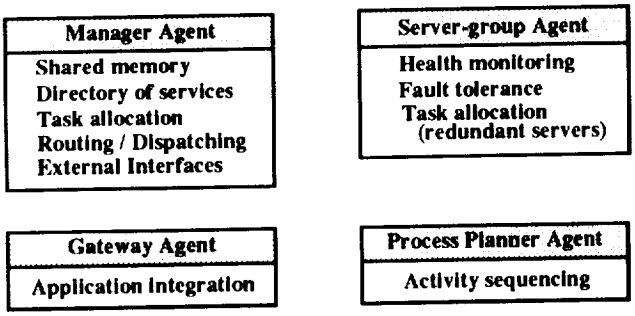


Figure 7: SOCIAL Library Building Blocks

The prototype distributed decision support system described earlier illustrates capabilities for:

- integrating independent planning and scheduling engines across a computer network;
- automating distributed interprocess communication, namely "fine-grained" exchanges of data and control between remote executing applications;
- automating conventional "coarse-grained" interactions such as file transfers across distributed application platforms;
- automating the coordination of complex sequences of fine- and coarse-grained interactions between distributed applications through high-level, declarative scripts.

The coordination capabilities provided by SOCIAL Manager Agents have broad applicability for distributed intelligent systems in space-related domains. For example, process scripts could be used to coordinate routine shop floor activities. Task control and work-in-progress status data could be routed automatically among Shuttle and payload processing facilities scattered across the Kennedy Space Center complex. Similarly, shop floor statistics could be collected, summarized, and transmitted to higher-level decision support systems. Mission schedules could be monitored and managed more effectively. This feedback could also be used to tune the processing estimates that drive long-term planning of Shuttle missions.

In addition, process scripts could be used to automate standardized launch processing and mission control disciplines, enhancing productivity, safety, and quality assurance. Beyond decision and operations support applications, process scripts can be used to automate routine flows of information in office automation and concurrent engineering contexts. Finally, process scripts can be applied in space science domains for automating sequences of data retrieval, analysis, and graphic visualization activities. End-users could develop, maintain, and extend their own application-specific scripts.

Acknowledgments

Development of SOCIAL has been sponsored by the NASA Kennedy Space Center under contracts NAS10-11606 and NAS10-11763. Artemis is a trademark of Metier Management Systems. Monte Zweben and Bob Gargan provided Gerry software. Brad Young provided technical assistance relating to Artemis.

REFERENCES

Adler, R.M. (1991a). A Hierarchical Distributed Control Model for Coordinating Intelligent Systems. *Proceedings of the 1991 Goddard Conference on Space Applications of Artificial Intelligence.* NASA CP-3103. pp. 183-198.

Adler, R.M. (1991b). Integrating CLIPS Applications into Heterogeneous Distributed Systems. *Proceedings of the Second CLIPS Users Conference.* NASA Johnson Space Center. Houston, Texas, September 23-25, 1991.

Adler, R.M. and Cottman, B.H. (1990). EXODUS: Integrating Intelligent Systems for Launch Operations Support. *Fourth Annual Workshop on Space Operations, Applications, and Research (SOAR-90).* NASA CP-3103, Volume 1. pp. 324-330.

Birman, K., Joseph, T., Kane, K., and Schmuck, F. (1989). *The ISIS System Manual V1.2.* Department of Computer Science, Cornell University, Ithaca, New York.

Bisiani, R., Alleva, F., Forin, A., Lerner, R., and Bauer, M. (1987). The Architecture of the Agora Environment. In M. Huhns (Ed.). *Distributed Artificial Intelligence.* Morgan-Kaufmann, San Mateo, California.

Bond, A.H., and Gasser, L. (1988). (Eds.) *Readings in Distributed Artificial Intelligence.* Morgan-Kaufmann, San Mateo, California.

Durfee, E.H., Lesser, V.R., and Corkill, D.D. (1989). Trends in Cooperative Distributed Problem Solving. *IEEE Transactions on Knowledge and Data Engineering.* 1(1), 63-83.

Gasser, L. and Huhns, M.N. (1989). (Eds.). *Distributed Artificial Intelligence, Vol. II.* Morgan-Kaufmann, San Mateo, California.

Gasser, L., Braganza, C., and Herman, N. (1987). MACE: A Flexible Testbed for Distributed AI Research. In M. Huhns (Ed.). *Distributed Artificial Intelligence.* Morgan-Kaufmann, San Mateo, California.

Hayes-Roth, F., Erman, L.D., Fouse, S., Lark, J.S., and Davidson, J. (1988). ABE: A Cooperative Operating System and Development Environment. In A.H. Bond and L. Gasser (Eds.). *Readings in Distributed Artificial Intelligence.* Morgan-Kaufmann, San Mateo, California.

Schantz, R., Thomas, R. and Bono, G. (1986). The Architecture of the Cronus Distributed Operating System. *Proceedings of the 6th International Conference on Distributed Computing Systems.*

Symbiotics, Inc. (1990, March). *Object-Oriented Heterogeneous Distributed Computing with Meta-Courier.* Technical Report, Cambridge, Massachusetts.

Zweben, M. and Gargan, R. (1990). The Ames-Lockheed Orbiter Processing Scheduling System. *Fourth Annual Workshop on Space Operations, Applications, and Research (SOAR-90).* NASA CP-3103, vol. 1. pp. 290-295.

Distributed Expert Systems for Ground and Space Applications

By: Brian Buckley, Interface & Control Systems

Louis Wheatcraft, Barrios Technology, Inc.

Abstract

The workstation, minicomputer, and microcomputer marketplaces have been revolutionized in the past decade by systems that are both open and distributed. As a leader in this revolution, the Naval Research Laboratory's (NRL) Naval Center for Space Technology (NCST), has been employing reusable software components to build a series of test beds, test and checkout systems for satellite assembly line operations, and distributed control of satellite tracking stations. The Navy has taken this one step further by unifying ground and space operations with the development of the Spacecraft Command Language (SCL).

SCL is a hybrid software environment borrowing from expert system technology, fifth generation language development, and multitasking operating system environments. SCL was developed by the Navy to be the controlling software for their Advanced Systems Controller (ASC). The ASC is a MIL-STD-1750A based Telemetry, Tracking, and Control (TT&C) controller for a new generation of Navy spacecraft having the capability of autonomous operation for up to 180 days.

Today's spacecraft are becoming increasingly more complex, with added sensors, higher data rates, and more capable standalone and distributed processors. The SCL system allows on-board processing of data, which has traditionally been considered to be in the realm of the ground segment. The distribution of processing to the space segment allows the spacecraft controller to analyze data points on-board and make decisions based on knowledge stored in the SCL scripts and rules.

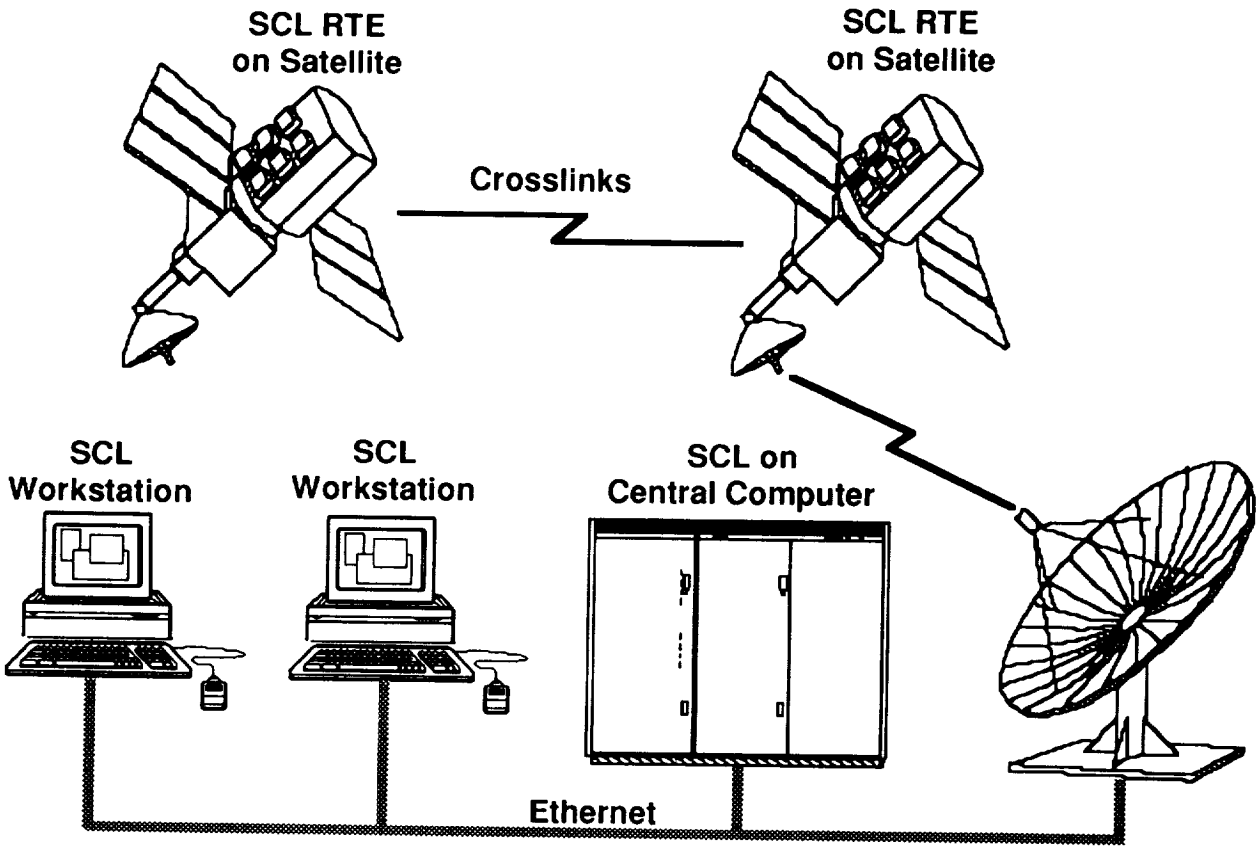
In addition, the spacecraft bus and payload systems are commonly developed independently, each having their own processors/controllers. Using a common distributed control language results in significant savings in total software development. The space-based SCL system can support distributed environments using a hierarchical scheme allowing subsystem controllers to communicate with a central controller.

A distributed approach is also used with the ground segment. Data points downlinked from the spacecraft are routed to workstations that analyze and view spacecraft and artificial telemetry points in real time. The workstation's knowledge base is used to analyze the telemetry and adjust the spacecraft's high level tasking to maintain the mission profile.

To unify the space and ground segments, the NCST has chosen the SCL system as the standard for use on-board the spacecraft as well as in the ground stations. The SCL system will run on a central ground station computer as well as on individual workstations used for subsystem monitoring and control. The SCL Real-Time Executive (RTE) on-board the spacecraft will be monitoring health and welfare, processing telemetry, scheduling mission tasking, and managing payload configuration changes.

Connectivity between multiple SCL nodes is not limited to exchanges of database items. A workstation can directly connect to any remote version of the SCL RTE. This allows direct control, interactive commanding, and real-time query of the remote SCL RTEs. This direct connect capability includes the version of the SCL RTE on-board the spacecraft.

This paper presents the SCL concept of the unification of ground and space operations using a distributed approach, describes the SCL system, offers examples of potential uses for the system,



Ground and Space Network using SCL

and details current distributed applications of SCL.

Introduction

The Naval Center for Space Technology (NCST) is in the process of developing the Advanced Systems Controller (ASC), which is a major upgrade to its current microprocessor based spacecraft controllers. The ASC hardware is based around the Honeywell GVSC MIL-STD-1750A processor and has been designed to be general purpose to allow tailoring of the system to meet the requirements of other spacecraft programs. The ASC software is based on the Spacecraft Command Language (SCL) Real-Time Executive (RTE). SCL is a hybrid system that employs a rule based event driven expert system as well as a procedural scripting capability.

The SCL development environment consists of a ground based windowed system used to develop SCL scripts and rules. The integrated environment consists of an editor, a compiler, decompiler, tracing subsystem, explanation subsystem, and the RTE. The SCL RTE was designed to be portable and run in a real-time embedded systems environment. The SCL RTE represents the majority of the code necessary to implement an embedded spacecraft controller. The NCST saw the need for the integration of ground and space operations with a common control system using a single control language. By using SCL in a distributed environment, the command language for ground and space segments share a common syntax. The SCL grammar is based on fifth generation languages and is very english-like, allowing non-programmers to write the scripts and rules that constitute the knowledge base. Because the ground-based SCL environment uses the same RTE as the spaceborne controller, scripts and rules can be developed and debugged on the ground-based RTE before requiring the spacecraft hardware for final checkout.

The NCST has incorporated a control system in its ground stations for the past decade. This control system has proliferated to integration and test environments, and to many of the supporting ground stations around the world. The SCL system has been integrated with this control system to provide additional, or "value-added" capabilities to the existing systems. Besides the goal of early deployment and checkout of the SCL software, the NCST felt that the existing ground stations could benefit from the expert systems capabilities provided by the SCL system.

The existing NCST satellites are well characterized and are managed by several software components and Orbital Operations Handbooks (OOH), which define configurations, constraints, and contingency plans. This knowledge can easily be translated into SCL's scripting language. The resulting knowledge base consists of SCL scripts, rules, and functions. The knowledge base is used in real time to monitor and detect changes in the vehicle configuration, maintain the configuration, move efficiently from one configuration to another, monitor system health, and perform command verification. At a ground station, copies of the SCL system are used on workstations to analyze telemetry and drive third party graphics products. The SCL workstations are able to advise an operator of anomalous conditions and suggest corrective measures, compare the current configuration against the desired mission tasking profile, and provide a capability to autonomously maintain vehicle configuration.

A New Way of Doing Business

In the past, only a ground-based command and telemetry database needed to be managed. With the advent of the ASC concept, the on-orbit SCL database must also be considered. The field sites throughout the world must also have knowledge of the database items that are on board, as well as the scripts and rules that are loaded on the ASC. All ground stations must have knowledge of the orbiting satellite's database.

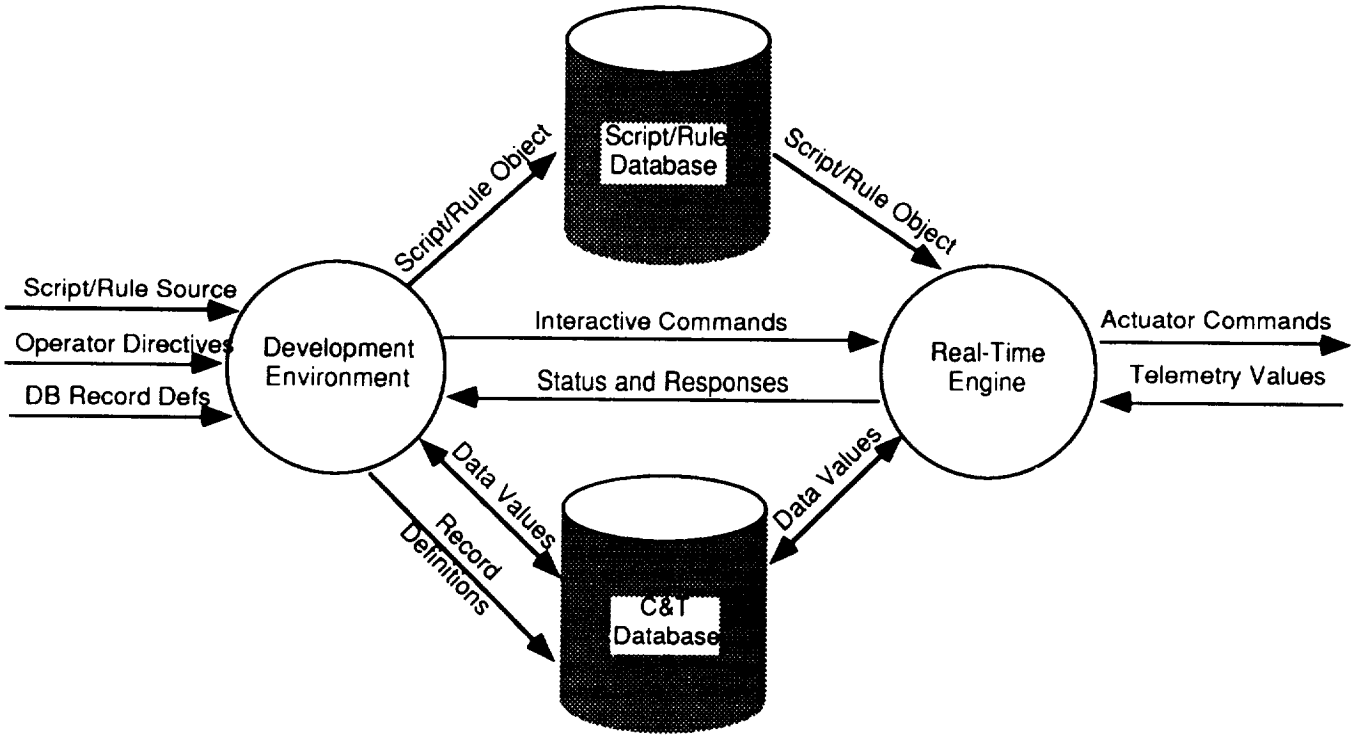
The current generation of spacecraft has a control system used for ground operations, an embedded control system for the spacecraft controller, and hard-coded algorithms for specialized hardware. Rather than use several different sets of software, the NCST approach is to use the same SCL software for spacecraft control functions as well as for ground station control. The SCL system is portable and has been designed to be used in embedded systems as well as workstations and minicomputers. This approach allows a common SCL grammar to be used for the ground station, the spacecraft controller, and payload controllers.

A major departure from the past and present spacecraft control systems, is the concept of using an existing, validated software "shell" for control system development. In the past, it was felt that each new spacecraft system needed a unique on-board software controller. Thus a special team of specialized programmers would develop a new control system from scratch. This proved to be a high risk as well as a very expensive approach both in terms of cost and schedule. With the SCL concept, the only unique software is the low level hardware interface code, the database, and the knowledge base. This approach has several advantages:

- The development cycle can be shortened; much of the code is "off the shelf".
- Risk is reduced. The SCL system has been proven both on ground and on space processors.
- The knowledge of a system is embedded in the controller.
- The learning curve for ground station operations is reduced since the knowledge is captured on the system.
- Consistent operations. Tasking of the vehicle is performed the same for both the ground and space segment.

The scripting contained in the knowledge base is written using a high level language that can be easily learned and understood by the subsystem engineers, thus not requiring a team of specialized programmers.

Mission tasking has traditionally been based on a time-line. At a given point in time the spacecraft is commanded to perform a function. Commanding can be carried out via stored commands, and by interactive commanding from the ground station. The time-line approach has proven to be cumbersome and difficult to administer. Previously, commanding was done "blind"; no database was available on the spacecraft for interrogation. With the availability



SCL System Dataflow

of SCL's scripting capabilities and flight GPS receivers and other equivalent devices, an on-board expert system that is performing real-time monitoring of the current spacecraft position can allow field of view tasking to be implemented. The spacecraft can collect data when an area of interest is in the field of view, and it can dump stored data when a field site is within its field of view. This approach can greatly simplify the mission tasking definition and the reduce the need for as many men in the loop.

The NCST envisioned using the SCL system on-board the spacecraft to share the testing burden, since it would be able to detect and isolate faults and report them to the ground. Time saved during the integration and test phase of the program can result in significant monetary savings. The capability to do self-diagnosis is desirable since a low earth orbiting spacecraft (LEOS) is in view of a ground station for only a small percentage of its orbit.

SCL System Architecture

The SCL system consists of five major components:

- The database describes digital and analog objects that represent spacecraft sensors and actuators. The latest data sample for each item is stored in the database. The database also contains derived items that are artificial telemetry items whose values are derived from physical sensors. Examples of derived items could be: average temperature, power based on current and voltage monitors, subsystem status variables, etc. Data structures required to support the Inference Engine are also stored in the database. These items include command actuators for commanding the spacecraft systems.
- The development environment is a window based application that includes an integrated editor, the SCL compiler, decompiler, cross-reference system, explanation subsystem, and filing system. The development environment is also used as a front-end to control the SCL RTE. A command window is used to provide

a command-line interface to the Real-Time Executive. Extensive use of pull down menus and dialogs are used to control the system.

- The RTE is the portable multi-tasking command interpreter and inference engine. This segment represents the core of the flight software. This portion of the software is available in both C and Ada to allow ease of porting to a specific hardware platform (ground or space).
- The Telemetry Reduction program is responsible for filtering acquired data, storing significant changes in the database, and presenting the changing data to the Inference Engine.
- The project is the collection of SCL scripts and rules that make up the knowledge base. On the ground based systems, the project contains an integrated filing system to manage the knowledge base. In the space environment, the binary knowledge base is uploaded to the spacecraft and stored in memory.

Depending on the needs of the user, all components of SCL can be run on a single system, or may be distributed among systems. The development environment can be used to directly connect to a local or remote version of the SCL RTE. This direct connect capability is also supported for the space segment to allow interactive commanding and query of the system.

Fielding the System

Due to the power and the advantages provided by SCL, the NCST decided to put the SCL system into the field immediately for several reasons:

- Risk Reduction - prove the system is viable through a series of proof of concept efforts.
- Capture knowledge of key personnel to allow the system to aid integration and test efforts.
- Allow parallel development of knowledge bases on workstations.
- Develop a concept for adaptive mission tasking, and field of view commanding.
- Allow development of simulations for Air Force and Navy Projects.
- Integrate SCL with existing systems to allow value-added features.

The SCL system was put through its paces early in its development cycle in a series of proof of concept efforts. The Lab Test Bed proof of concept used the SCL RTE on a UNIX platform to control a prototype satellite. To demonstrate the enhanced capabilities of SCL, the following demonstrations were performed:

- SCL Satellite Configuration and Auto reconfiguration. The goal of this demonstration was to show SCL is capable of:
  - Transitioning from one control language to another. This was demonstrated by translating existing control system command procedures into SCL scripts.
  - Commanding the spacecraft and receiving telemetry responses.
  - Detecting that the spacecraft is not in a desired configuration and notifying the operator.
  - Automatically reconfiguring the spacecraft to a safe state in event of an error.
- SCL Commanding. The goal of this demonstration was to show several SCL capabilities:
  - Simple commanding and verification by monitoring telemetry points to verify that the command was successful.
  - Mission constraint checking by verifying telemetry prior to a command being issued to prohibit a potentially damaging command from being sent.
  - Abstract commanding by using SCL's high level commanding capabilities. Scripts are used to check telemetry and manage primary and redundant sides of boxes, and allow a default side to be active.
- Fault Tolerant Configuration. The goal of this demonstration was to show the SCL capability to react in real time to telemetry changes and implement an alternate course of action if conditions warrant. This demonstration used redundant sides when the primary side did not respond.

```
constraint    HOT_SWITCH
subsystem     TRANSMITTER
category      SWITCHING
priority      15
activation    YES

   if
      (BIU_CROSS or BIU_NORM)
   and
      XMIT_POWER = ON
   then
      reject
      execute fault_log with constraint_err
   end if
end   HOT_SWITCH
```

SCL Constraint Example

- Field of View Operation. The spacecraft position coordinates are telemetry database items and may have rules associated with them. When a significant change in the position data occurs, the rules associated with them are executed. The rate that the position is updated is determined by the desired ground track accuracy. The field of view can be calculated from the position coordinates and compared to the area of interest. If the area of interest is within the field of view, the rule may execute scripts or sequences of commands to change the spacecraft configuration.

- Mission Tasking. The goal of this demonstration was to highlight three tasking aspects of SCL. First, SCL is required, at a minimum, to reproduce the current capabilities to schedule and activate various configurations. Second, SCL is capable of a variety of common cyclic functions that can be scheduled within the SCL kernel. Third, SCL is capable of multitasking. This multitasking capability will reduce the time, effort, and complexity of resolving the varied resource needs between program entities. By supporting multitasking, SCL can satisfy requirements from many different sources.

- Self-Testing. The NCST has designed the ASC with a goal of improving testability. By using the ASC processor as an asset for testing, parallel testing can be accomplished. Having an intelligent controller allows the system to perform self-diagnosis, trouble shooting and reporting of problems. SCL enhances testing by providing a flexible and re-usable means of implementing on-board testing. At the subsystem level, scripts and rules can be written to provide a test environment for a specific subsystem. SCL can send commands to the unit, and react to the telemetry responses from the unit. This would provide a common test method for many layers of the system, allowing consistent testing throughout the phases of system integration. At this level, SCL supports command and telemetry verification for each box.

```
-- This script prepares the Reaction Control
-- Subsystem for a thruster firing and calls a
-- subroutine script to perform the actual
-- firing. It accepts two parameters: one
-- indicating which thruster to use and
-- another specifying the duration of the firing

const  fore 1     -- define a constant for the

                  -- forward thruster

script Maneuver1     thruster, duration

   -- command to enable thrusting
   set RCS_ENABLE to ON
   -- allow propellant flow
   set TANK_ISO_VALVE to OPEN

   if
      thruster = fore
   then
      -- call Fore thruster subroutine

      execute ManvFore with duration

   else
      -- call Aft thruster subroutine
      execute ManvAft with duration

   end if

   -- command to disable thrusting
   set RCS_ENABLE to OFF
   -- command to isolate tank
   set TANK_ISO_VALVE to Closed

end    Maneuver1
```

SCL Script Example - Mission Tasking

- Simulation. Frequently, in the production of the spacecraft, there are subsystem components that have not been integrated or are missing due to troubleshooting or modification. In their absence, test procedures have to be modified or must be postponed until the component is available. Having a simulator for a missing component is desirable so test procedures can be run without modification and testing can proceed without the complete system in place. In the event a particular box is absent from the spacecraft, its presence could be simulated by the SCL inference engine, either operating in its embedded form on-board the spacecraft, or in its ground system form operating on the ground station processor. To simulate a missing component, a knowledge base must be developed to respond to commands defined for the component. When a command is sent to the component, the associated rule is executed and a corresponding telemetry response is generated.

```
rule        BATT3_TEMP
subsystem   EPS
category    BATT3
priority    15
activation  YES

    if
        BATT3IT > 50
    then
        set alarmlevel of BATT3IT to RED
        execute battsafing with 3, priority = 30
    end if
end     BATT3_TEMP
```

SCL Rule Example

Further Proving of the System

In another proof of concept, the NCST wanted to test the expert system technology in a "real-world" scenario. This proof of concept required that the SCL system be compared to a commercial off the shelf (COTS) expert system. Both SCL and the COTS system were required to be capable of being used in embedded systems (i.e., blown in PROM). The expert systems were to be used to implement the flight algorithms for NRL's upper stage used for orbital insertion of satellites. The upper stage is spin stabilized until it reaches the insertion orbit. Once in the desired orbit, the upper stage is spun down and stabilized using momentum wheels and reaction control thrusters. The upper stage then jettisons the spacecraft allowing it to move into its parking orbit.

All aspects of the orbital transfer maneuver are controlled by the Attitude Control Electronics (ACE). The ACE subsystem is semi-autonomous and can issue thruster commands to maintain the desired attitude. The ACE control loops were developed in the flight processor's native assembly language. The development of the algorithms required years of design, testing and elaborate simulation. Within 3 months, two prototypes were generated using SCL and the COTS expert system. The two prototypes were exercised using the same flight qualification test used for acceptance testing of the original ACE flight software.

The results of this effort proved that the COTS expert system was NOT able to keep pace with the flight control loops, resulting in additional thruster burns to stabilize the spacecraft. The knowledge base for the COTS expert system was re-designed, but was still unable to keep up with the control loops. The SCL system however, performed the ACE algorithms as efficiently as the flight software.

Based upon the successful demonstrations of the SCL system, the NCST baselined the SCL software as the control system for the Advanced Satellite Controller. The system has also been chosen as the control system for two NASA projects, one of which will launch in September of 1993.

Satellite Simulations

In the summer of 1990, the NCST was chosen to provide spacecraft simulations for the Space Defense Initiative Office (SDIO) Standard Mobile Segment program. The NCST chose to use the SCL system to provide command response capabilities and electrical and thermal modeling for the FLEETSATCOM and GPS

satellites. An SCL rulebase was developed to decode binary commands and insert an appropriate response in the telemetry bit stream. Telemetry from this bit stream was distributed over Ethernet to Air Force contractor workstations. The SCL system was integrated with the NCST's TT&C system to allow real-time simulation of command responses. The electrical and thermal models were also developed as part of the SCL knowledge base. These models provided a detailed emulation of the spacecraft in real-time or up to 600 times real-time.

The SCL simulations were developed on workstations and delivered on the host computer as a stand-alone entity. The system was activated from the TT&C system, and runs with little or no operator intervention. The only intervention required is when an operator wishes to generate anomalies in a scenario. These simulations have been delivered to Air Force contractors, to the National Test Bed, and to the NAVSOC facility at Pt. Mugu, California. The NAVSOC personnel currently use the simulators for FLEETSATCOM training. In the near future, the SCL software will be embedded in a high-fidelity hardware simulation for a NCST program.

Integration with Existing Systems

Currently, the SCL system is completing its integration with the NCST's ground station software where it will become part of a client-server model. The SCL system will reside on several workstations as well as the ground station's central computer. All communications will be through a packetized message passing protocol over Ethernet. The ground station TT&C software is responsible for telemetry decommutation and distribution. The SCL workstations will monitor appropriate telemetry to generate operator advisories and drive graphics interfaces, which are used to indicate the spacecraft configuration, health, and welfare.

The NCST tracking stations are taking advantage of the distributed aspects of SCL using a network of minicomputers and workstations. Once the NCST's ASC is launched, the full potential of SCL can be exploited. In addition to the ground-based network, the spaceborne ASC platforms will be capable of communicating with each other. Ground stations will be able to perform the central site load to one ASC. The ASC that is loaded will be capable of forwarding the applicable script, rule, and database loads to the other ASC's. Since the ASC's can be in constant communication with each other, the mission tasking load can be balanced among the cluster of ASC's. This concept of adaptive tasking will be managed by the on-board SCL expert systems. Each SCL knowledge base will know the configuration of the on-board ASC, and can query the other ASC's to obtain their current configuration and tasking profile. Having this capability will create a network of ground and spaceborne SCL platforms. With the ability to upload databases and knowledge bases, the possibilities for this network are tremendous.

Reusable Controllers

Recently, SCL was chosen for a commercialization of space contract funded by NASA Goddard Space Flight Center. The Autonomous Rendezvous and Docking (ARD) satellites will use SCL to control docking and fluid transfer experiments. The ARD satellites will use off-the-shelf spacecraft computers based on the 80186 chipset. The ARD satellites will be low earth orbit satellites. The ground stations will use SCL to monitor telemetry and send commands. The two satellites will both be controlled by an embedded version of SCL and will communicate with each other during the docking procedure through RF modems. When the satellites are within a kilometer of each other, one satellite will act as master, and the other as slave. The SCL knowledge base on one platform will be sending commands to control the maneuvers of the other.

The same off-the-self spacecraft controller will be used for a material processing experiment to be launched as a NASA Get Away Special (GAS) on-board the Space Shuttle. This captive experiment will use SCL to control an oven and a robot that will be used to place material samples in an oven to test the effects of annealing in a weightless environment.

Lessons Learned

Development of a distributed expert system for ground and space did not come without its share of technical and psychological obstacles. The

following paragraphs give an overview of some of our challenges.

Portability: The SCL system was originally developed on a Macintosh II platform. The Macintosh proved to be a highly productive environment because of its integrated toolkit for windowing, the operating system, and the filing system. The software development tools available on the Macintosh were the most affordable and most sophisticated at the time. We made great strides in the development of the systems, but we were faced with the chore of porting the system to other platforms. The NCST tracking station uses the Digital Equipment Corporation (DEC) VAX family of computers and workstations running the VMS operating system. The SCL code was originally written in C, and we had to convert the real-time engine and database loader to ANSI compatible C. We also needed to support UNIX platforms and IBM PC platforms. This required adding conditional compilation statements for some of the include files since paths are different.

To keep the core software identical on all platforms, the operating system specifics and the I/O have been abstracted to a very small number of routines, which are replaced on each system. These routines interface directly to the host operating system to schedule execution, map memory sections, and obtain systems time. The low level I/O and network I/O is also handled in this group of routines. This abstraction of I/O has allowed the system to be easily ported to multiple hardware platforms, operating systems, and real-time executives for embedded systems.

Another major obstacle was communication between local and remote versions of SCL on a non-homogeneous network. Different machines were either big-endian or little-endian (high byte then low byte in memory, or vice-versa); they also use different floating point formats. The low-level I/O modules were modified to determine the "sex" of the local and remote SCL systems and perform any data transformations necessary. The network I/O has been sufficiently abstracted to allow communication via TCP/IP and DECnet protocols over Ethernet, Appletalk, serial communications such as RS-232, and custom protocols.

We felt it was prudent to allow the spaceborne, embedded version of SCL to perform native access to all data structures including the database, and the knowledge base. To allow the embedded SCL to have native access, the ground based development environment had to support a cross-compilation of all data destined for a remote version of SCL. By setting a software switch, the data streams, and files produced, are formatted in the target processor's native data structures. The development environment is also capable of decompiling the data streams from the target platform.

Necessity vs. "Feature-itis": As the system evolved, new features were added as needs and requirements dictated. At one point we found ourselves adding features because we thought it would make the system "slick". As we found out, new features and new code created side effects that were not discovered without extensive testing. We also found that many of the "slick" features were difficult to duplicate on other platforms, since they did not have an integrated toolkit like the Macintosh. We were striving to maintain a common look and feel for the product across platforms. Because of the porting of the code to multiple platforms, the system was baselined (frozen) and only changed for maintenance and bug fixes.

The SCL proofs-of-concept resulted in another company providing an objective analysis of our product. Our system was compared with commercial products to test functionality as well as real-time performance. As a result of the comparisons, we added several extensions to the grammar, and an additional user-selectable, inferencing strategy. Other behavioral quirks were also corrected. We did not however, add more object oriented features due to the real-time considerations. The SCL system is designed for real-time embedded environments and pre-allocates all data structures prior to startup. The SCL RTE does not perform any dynamic memory allocation due to memory fragmentation issues. Traversing the data structures necessary to implement additional object-oriented features would degrade the real-time performance and increase the memory requirements for the system.

Information Management: For past programs, the spacecraft controller used a low-level command-language and did not support an on-orbit database. The ground station and test systems were the users of a database. With the introduction of the ASC (with SCL on-board), the ground sites as well as the spacecraft must

contain copies of the database. Additionally, a core set of scripts and rules must also be managed. In the past, it has been difficult to ensure that all mission sites contain a database which describes the same command and telemetry points as the central site. The prime contractor is now responsible for delivering and configuration managing databases for each site and platform. Distribution of the databases are from a master database at a central site. All other sites' databases are derived as a subset of the central site's database. The spacecraft data points are the same from site to site, but the databases at each site can be extended to include ground specific data points.

Currently the SCL development environment compiles scripts, rules and database records and assigns ID's to each. Scripts, rules, and database records are referenced by these ID's. To keep all sites synchronized, a given ID must correspond to the same script, rule or data point at each site. The situation is further complicated when the user references data points or scripts on another platform or network node. Several options are available to guarantee that an ID is unique among all nodes and platforms in the network. The strongest contender as a solution is to use a hash algorithm to define the node and script/rule/database item combination. This scheme results in a 64-bit ID for each object and doubles the current size of the ID in the SCL intermediate code. In addition to the extra data word requirement, several table lookups are required to efficiently look up the address for the data structures.

Another area of concern was how to distribute the calculation of the artificial data points (derived items). It was felt that the prudent approach was to divide the derived item calculations between ground and space. Rules are used to calculate derived items and are defined on the appropriate platform. Spaceborne derived items might be used in calculations for attitude control, where derived items would be used on the ground to drive graphics displays. The ground derived items are further distributed to workstations that analyze telemetry for specific subsystems.

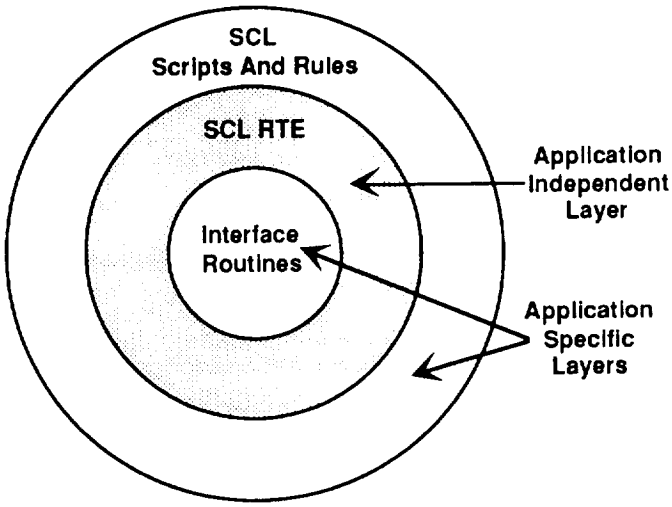
Windowing & Reusable Code: As prototypes of the SCL system were ported to other platforms, the amount of code was increasing rapidly. This problem was complicated by the fact that the windowing systems on each platform (Macintosh, OSF/Motif, Microsoft Windows) behaved quite differently from a programming viewpoint. We estimated that a man-year would be required to bring a programmer up to speed on each windowing system to generate a production quality application. We also saw that many software functions were being replicated even within the same programming team. It was at this point we decided to port the SCL development environment over to C++ to promote code reusability and abstraction from the windowing system specifics.

We saw four layers of class libraries that needed to be defined. The foundation of the system is the portable filing system layer and the database management layer. The filing system layer implements a class library modeled after the Macintosh resource files. The filing system is based upon a four character ASCII key that is used as an index. Each key (or resource) can contain variable length data structures identified by a unique name and index. This filing system is used to maintain a consistent interface across all platforms.

The data management layer is based largely upon the public domain National Institutes of Health (NIH) class library. This class library manages commonly used data structures: lists, sorted lists, data blocks, strings, collections of objects, etc. This layer is a simplified subset of the functions contained in the NIH class library. Data persistence is implemented to allow data structures to be maintained in memory once they have been read from disk. This has significant performance advantages since the disk is only accessed again when the file is closed, or the programmer explicitly requests that the data be written back to disk.

Abstraction: The SCL system was envisioned from its inception to be applicable to other types of controllers and other satellite programs. To accomplish this, the system had to abstract the specifics of the application from the knowledge engineer. The grammar for SCL is a hyper-scripting language that supports object-oriented features. The object-oriented approach allows the Real-Time Executive to treat Actuators, Sensors, and Derived Items essentially the same way. All decisions as to how to perform the I/O is deferred to the lowest level interface routines. These routines are the "glue" between the logical and physical interfaces. This approach allows a few

hundred lines of code to perform any transformation required by the hardware interface. This approach allows identical scripts to run both on workstations and the flight processor. Both implement the same functionality, but one could communicate over Ethernet, while the other communicated with a TT&C bus, or an I/O card in the same chassis.



SCL Software Architecture

Abstraction is also the key to keeping the vast majority of the SCL code portable. All operating system specifics are isolated from the code in just a few routines. These routines are replaced on the target machine with calls to systems services specific to that operating system.

Fear Of Artificial Intelligence/Expert Systems: Perhaps the largest hurdle to overcome is an inherent fear or apprehension of managers that they do not want to lose control of their spacecraft to a computer. They simply don't want to accept the perceived risk, and are more comfortable with the old or existing methods. To overcome some of the initial negative reactions, we have had to avoid the terms artificial intelligence and expert systems. Instead we use the term "Smart Control System". There is probably a better term. The main points that must be made are:

- Existing methods of ground up development are just re-inventing the wheel and are more risky because of the use of "new" and unproven code. They are also more costly because of the time to develop a controller from scratch and because of the increased schedule time.
- Rules are already embedded (hard coded) into existing software. But because they are coded by specialized programmers, it is difficult for the subsystem engineers to review and understand how their systems are being monitored and controlled.
- As stated previously, the SCL system employes the concept of using an existing validated software "shell" for control system development. With the SCL concept, the only unique software is the low level hardware interface code, the database, and the knowledge base. The scripting contained in the knowledge base is written using a high level language that can be easily learned and understood by the subsystem engineers, thus not requiring a team of specialized programmers. The rules that exist in the traditional controllers are now structured as individual items that are evaluated by the inference engine. This structure makes it easy for the subsystem engineers to review and understand the how their subsystem is being monitored and controlled.
- The amount of control given to the SCL system can vary depending on the needs and the configuration of the system. SCL can be used to duplicate an existing system's capabilities, perform data pre-processing, self-test, or autonomous control. The amount of control given to the system can be determined by the project office.

Expert systems have been recognized as being applicable to ground and space applications. However, association with Artificial Intelligence continues to project negative connotations for some people; the relationship must often be avoided.

Other Uses for SCL

The use of abstraction and object-oriented techniques allows the SCL Real-Time Executive to be applied in a variety of areas:

Spacecraft controllers: The RTE is available in both C and Ada making it applicable to a wide variety of processors.

Subsystems Controllers: The SCL system is designed for distributed environments and as such allows for a hierarchal bus structure for subsystem controllers to report to a system controller.

Centralized ground station computers: The system can be used in conjunction with X-Terminals to manage ground station resources such as antennas, frame syncs, command encoders, etc. The system can also be used for scheduling ground station resources.

Workstations: The SCL system is ideally suited for use on workstations to allow distributed processing and parallel analysis. The system is also useful for driving graphics and visualization tools. Results from local workstations can be reported back to a central computer, or commands may be uplinked to change or correct the mission profile. SCL has been demonstrated to be capable of analyzing the spacecraft configuration and providing advisories for operators and spacecraft engineers.

Integration and Test: The SCL system is used at Integration and Test (I&T) facilities to perform automated command procedures. The procedures developed at the I&T facility can then easily migrate to the tracking facility. If SCL is also used for the spacecraft, the scripts and rules can also migrate to the spacecraft. The on-board processing capabilities of SCL allow a spacecraft to perform self-health diagnostics and report its status to the ground.

Test Equipment: Quite often, mission unique hardware must be developed for test equipment. The equipment often requires a control system and a language to allow it to be commanded to perform its functions. SCL can easily be embedded in the target hardware to provide a standard interface for all phases of testing.

Simulation: SCL has been proven to be capable of providing command response simulations as well as detailed modeling of systems. The simulations can be used to augment missing subsystems during spacecraft integration as well as provide a common platform for system training of operators and other personnel.

Conclusions

The SCL system is quite feasible for use on distributed systems for ground and space. The SCL system also helps promote a standard interface for the many facets of ground and space. The system does introduce information management problems that are overcome by a disciplined approach to configuration management. This disciplined approach must also extend to the distribution of databases and knowledge bases. The system is several years into its development, has had numerous proofs-of-concept, and is in use at several sites. The SCL system provides a low-cost, low-risk solution for many of today's command and control environments.

Acknowledgments

We would like to thank the following people for their contributions to this paper: Dave Schriftman, and Patrick Pinchera.

References:

1. Buckley, Brian and Wheatcraft, Louis: *Spacecraft Attitude Control using a Smart Control System*, SOAR Symposium, Houston TX., July 1991

2. Buckley, Brian and Wheatcraft, Louis: *Spacecraft Command Language - A Smart Control System*, Interface and Control Systems, Melbourne, Fl., Barrios Technology, Houston, TX., March 1991

3. Van Gaasbeck, James: *Technical Overview of the Spacecraft Command Language* Naval Research Laboratory, Washington D.C., 1991

4. Interface and Control Systems: *SCL User's Guide*, Naval Research Laboratory, Washington D.C., 1990

5. Buckley, Brian and Wheatcraft, Louis: *Rapid Prototyping of a Spacecraft Controller*, JAIPCC Symposium, Houston TX., March 1991

6. Buckley, Brian and Wheatcraft, Louis: *Spacecraft Simulations with a re-usable Smart Control System*, JAIPCC Symposium, Houston TX., March 1991

Evaluating Model Accuracy for Model-Based Reasoning

Steve Chien and Joseph Roden
Jet Propulsion Laboratory, California Institute of Technology
Pasadena, CA

Abstract

Model-based reasoning has been proposed as a general methodology for such diverse tasks as monitoring, diagnosis, control, and design. In this approach, a behavioral model mimicking the structure of the target system is used to reason about expected performance of the target system. However, most such work does not explicitly account for inaccuracies in the model.

This paper describes an approach to automatically assessing the accuracy of various components of a model. In this approach, actual data from operation of the target system is used to drive statistical measures to evaluate the prediction accuracy of various portions of the model. We describe how these statistical measures of model accuracy can be used in model-based reasoning for monitoring and design. We then describe application of these techniques to monitoring and design of the water recovery system of the Environmental Control and Life Support System (ECLSS) of Space Station Freedom.

Keywords: model-based monitoring, diagnosis, control and design, validation of knowledge-based systems, model-based simulation

1. Introduction

Model based reasoning has been advocated as a general approach to a wide variety of tasks such as monitoring [Doyle et al. 89, Doyle et al. 91, Dvorak & Kuipers 89], diagnosis and interpretation [Davis and Hamscher 88], control [Scarl et al. 88], and design [Chien et al. 91a, Chien et al. 91b, Bose & Rajamoney 91]. However, despite this strong effort, comparatively little work has focused upon using actual data on model performance to characterize how well a model captures the behavior of the target system.

This paper describes a statistical approach to measuring the prediction error of a model based upon an analysis of model prediction performance on actual data. This analysis produces a statistical model of expected model prediction error. This model of the model error is then used in the model-based reasoning tasks of monitoring and design.

The next section of this paper describes how the statistical techniques are used to create a model of the error and how this model of the error can be used to calculate confidence intervals. The following section describes how this confidence interval information can be used in model-based monitoring and design tasks. This section also describes several applications of this error model to monitoring and design of the Environmental Control and Life Support System for Space Station Freedom. The discussion section of this paper focuses upon ongoing work to increase the accuracy of the error models by applying machine learning techniques to learn error models.

2. Evaluating Model Accuracy

Model-based reasoning uses a model of a system to predict the behavior of the system under the conditions included in the scope of the model. It is useful for applications using a model to know how accurately the model predicts the behavior of a system being modeled. Instead of simply predicting that a measure will take on some value, it is more useful to state how confident the model is in predicting that value.

Authors' Address: JPL, M/S 525-3660,
4800 Oak Grove Dr., Pasadena, CA 91109-8099
Internet: {chien,roden}@aig.jpl.nasa.gov

Model accuracy can be evaluated by comparing model behavior to observed system behavior. Through analysis of errors in predicting system behavior, we can estimate the amount of error we expect a model to produce. Data obtained by performing model evaluation studies can provide a basis on which to model the errors a model produces.

Model error (Δ) is defined as the difference of the model predicted value (m) based on previous observed system values and the current observed system value (o) for a given system state:

(1) $\Delta = m - o$

For our applications, the time step is relatively constant. Thus the model is making a prediction of the i-th time step from the data of the (i-1)th time step. In Figure 1, the model error is the difference between the model predicted value and the observed system value over time.

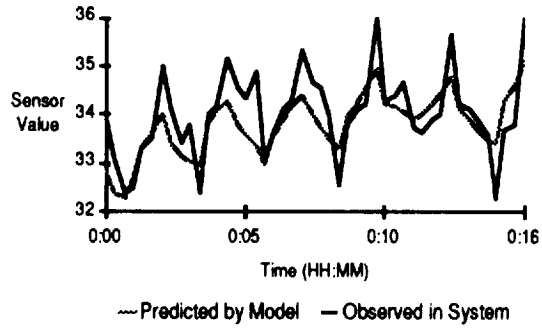


Fig 1. Graph showing difference between model predicted value and observed system value over time

For a given operating mode of a system, some number (n) of observations of one time step model predictions are taken, and the error computed for each. This results in a sampled model error distribution of Δ_1, Δ_2, ..., Δ_n. From this distribution we develop a general model of the error. We observed our samples to be approximately normally distributed. Figure 2 shows a histogram of model error for a particular sensor. Given this sampled error distribution, we estimate that the true model error is normally distributed with mean $\overline{X}$ and variance s^2.

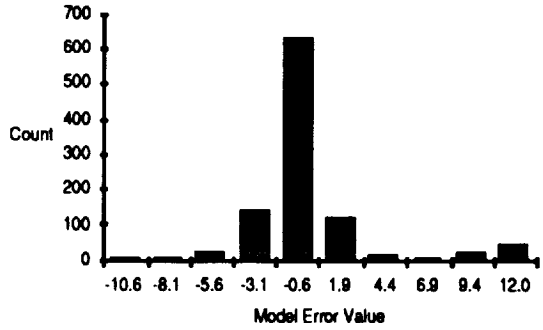


Fig 2. Histogram showing model error distribution for sensor KP02

For a given operating mode of a system, we evaluate model accuracy by determining the probability (p) that the true system measurement (t) will take on a value within a range of values (m-e, m+e) around the model predicted value (m). This probability can be determined given the distribution of the model error. Alternatively, by setting an acceptable probability bound, we can determine the range of values that the true system will take on with that probability. Equation 2 shows how this probability is determined given Φ, the cumulative distribution function of the standard normal distribution.

(2) $P(m-e \le t \le m+e) =$

$$\Phi\left(\frac{e - X}{s}\right) - \Phi\left(\frac{-e - X}{s}\right)$$

Using the probability table for the standard normal distribution, this measure quantifies the accuracy with which a model predicts the true system measurement.

3. Applying the Error Model in Model-based Reasoning

This section describes two applications of confidence intervals to model-based reasoning: model-based monitoring as applied to discrepancy detection and model-based prediction for design.

3.1 Application to Model-based Monitoring

One application of model accuracy is in model-based monitoring [Doyle et al. 89, Dvorak & Kuipers 89]. In model-based monitoring, a model of the target system is used to predict sensor values. Deviations from the predicted values are indications of abnormal behavior and are thus indicative of sensors which should be reported to operators. However, if portions of the model in certain operating modes are inherently inaccurate because of noise or poor understanding or predictability of the occurring phenomenon, the strength of the model's predictions should be correspondingly reduced.

One method to account for model inaccuracy in model-based monitoring is use of a running average of model/actual deviation [Doyle et al. 91]. In this approach, a running average of the deviation between model predicted and actual values is maintained. By tracking the current deviation minus the running average of the deviation, the current deviation can be ignored in cases where the model has not been tracking the system behavior accurately.

While the deviation from running average deviation measures the recent performance of the model, statistical measures over all available historical data provide a measure of past historical performance. With a confidence interval capability as described in the previous section, a more direct approach to calibrating deviation scores according to model accuracy can be applied. Specifically, the statistical model of the prediction error of the model can be used to generate a measure of the unusualness of a deviation of the model from the observed value.

For example, consider the following example from our ECLSS monitoring application. Using the techniques described in the previous section, a sensor KP02 is measured to have a error with a measured distribution of mean -0.19 and standard deviation 1.89 in system operating mode PROCESS. If we observe a discrepancy of 3 PSIG between the observed and predicted values, we can now use the equations shown in Section 2 to produce a confidence rating of 0.89 that the model is within 3 PSIG of the actual. Thus high confidence values for the error being less than the current deviation indicate unusual deviations.

As a second application to model-based monitoring, consider the case where a model historically predicts well but has recently been predicting poorly. This may indicate a persistent unexplained phenomenon affecting the sensor or portion of the system in question. Such a situation could be detected by determining if the running average of the deviation is at a level which is relatively unusual given the error model (e.g. for a running average deviation e, $P(-e \leq \Delta \leq e)$ is high). This provides a measure of the unusualness of the running average of the deviation. Note that this monitoring measure and the previous one are complementary. The unusualness of the current deviation catches quickly developing departures from normal operations, but is susceptible to random noise. The unusualness of the running average of the deviation is not susceptible to random noise, but takes longer to manifest and inform.

3.2 Application to Model-based Prediction for Design

Another application of model-based reasoning is in evaluating sensor placements for assistance in the design process. Specifically, we have been working on approaches to evaluate sensor placements with respect to a diagnosability criterion [Chien et al. 91]. In this approach a model of the target system is used to determine how specific proposed sensors would report altered scores in the event of a fault occurrence. More specifically, we evaluate how well a sensor can distinguish between classes of states with respect to three criteria. For the purposes of fault detection, the relevant distinction is between faulted and non-faulted states. For the purposes of fault isolation, the relevant distinction is between faulted states.

Towards evaluating diagnosability, we have developed three measures. First, *Discriminability* measures how much of a

divergence the model predicts would occur in comparing between the two states. Second, *Accuracy* measures the confidence in the model's prediction of the expected divergence. Third, *Timeliness* measures the time lag between the occurrence of the fault and the discrimination detected by the sensor.

Using the measure for model error we have described in this paper, we can formulate the confidence that the predicted divergence would be predicted and the actual value not deviate as a probability. The lower the probability of this occurrence, which represents the model predicting a change that does not occur, the more likely the sensor will be able to perform the discrimination.

3.3 Examples from Application to the ECLSS Testbed

Our sensor placement approach is being tested upon the water reclamation subsystem of the Environmental Control and Life Support System (ECLSS) for Space Station Freedom. A model describing the behavior of the Multifiltration Subsystem (MF) in terms of fluid flow and heat transfer has been constructed. This model was developed via a combination of study of design documentation (i.e., schematics, etc.) and consultation with domain experts (e.g. the operators of the testbed). This model has been validated by comparison against actual data from the subsystem testbed undergoing evaluation at the Marshall Space Flight Center in Huntsville, Alabama. We also have constructed models of the Vapor Compression and Distillation (VCD) and Volatile Removal Assembly (VRA) subsystems of SSF ECLSS. Together, these models represent coverage of virtually the entire water-side of SSF ECLSS. We are also in the process of extending our model to cover ECLSS air-side subsystems.

Figure 3 below shows the ECLSS multifiltration subsystem. In this subsystem, the water first passes through a pump at the inlet to the MF system. Next, the water passes through a coarse filter before entering the sterilization loop. In the sterilization loop the water is heated in the regenerative heat exchanger and then by the in-line heater after point 3. Within the sterilizer reservoir, the temperature of the water is maintained at 250° F for several minutes. In the second portion of the subsystem, the water passes through a set of unibed filters designed to remove particulate contaminants from the water. Possible sensor types are flow rate, water pressure, and temperature. Possible sensor locations are indicated by ovals in Figure 3.

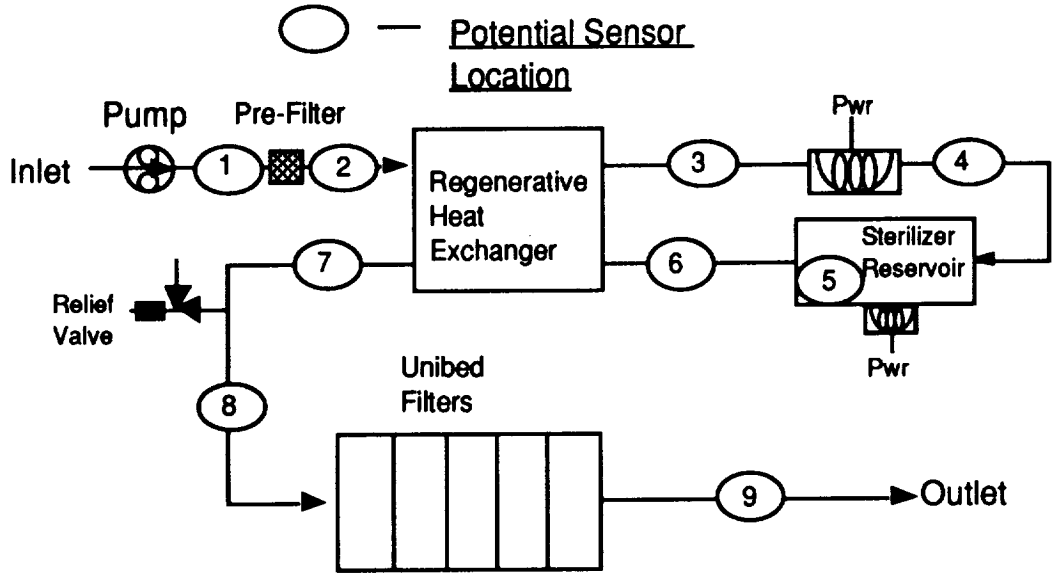


Fig 3. The Multifiltration Subsystem

In model-based monitoring, the empirically derived model accuracy scores impact monitoring in the following way. The process mode model of the conductivity sensor KT02 at point 4 exhibits poor model accuracy (empirically derived mean of 2.15 and standard deviation of 27.24). Thus, relatively large deviations from model predicted values such as 4, with a confidence rating of 0.52, do not cause the sensor score to be brought to the attention of the operator. However, the process mode model of sensor KP02 is more accurate (mean -0.19 and standard deviation 1.89) so that relatively small deviations on the order of 4, with a confidence rating of 0.97, cause the sensor to be flagged and the sensor value to be brought to the attention of the operator.

In model-based diagnosability assessment, again the model accuracy figures heavily in evaluating certain sensors. For example, one possible fault is unibed loading, which occurs when particulate matter gets caught in the unibed filters. This fault has several effects. First, a pressure drop would occur, causing a lower pressure at location 9. Second, unibed performance would decrease, resulting in an increase in conductivity downstream from the unibeds. Third, if loading is significant, flow in the entire subsystem may decrease. Again, because the conductivity models are not very accurate, the *Accuracy* measure of the diagnosability evaluation would score the pressure sensor placement higher than the conductivity sensor placement for fault detection of this fault.

4. Discussion

This work is preliminary; there are a number of outstanding issues. One issue is the selection of a normal distribution to model the error. Other possible distributions may model the error more accurately. A measure of how well the derived error model matches the observed distribution would be useful in assessing the degree of confidence in the error model.

Another issue is our choice to model the error in absolute terms rather than as a percentage bound based upon the current model prediction (e.g. 4 PSIG ± 5% rather than 4 PSIG ± .20). This has ramifications if the model error tends to increase as a function of the model predicted value. A cursory analysis indicates that in general, in our domain, the error is not strongly correlated with the model predicted value so that modelling the absolute error seems reasonable.

We also model the model error independent of potentially relevant factors such as other causally related model predicted values. For example, the model may use an equation to derive a temperature in the MF subsystem that is accurate only in cases where the water pressure is high. One extension of our work focuses upon using machine learning techniques to determine what other potentially relevant factors would be good indicators of model accuracy. In this work we are investigating applying GID3* to learn a model accuracy function for a sensor S based upon model predicted value for S and other sensors.

Another outstanding issue is that of dealing with variable time steps. The accuracy of the model's predictions clearly depends upon how far into the future the model is required to make predictions. Currently, our model of the model prediction error does not account for this variable.

5. Conclusion

This paper has described an approach to evaluating the accuracy of a model's predictions. This approach uses statistical methods to develop a model of expected error in model predictions. This paper has also described how this statistical measure for model error can be used in two model-based reasoning tasks: model-based monitoring and model-based reasoning for evaluating sensor placements. By application of our derived measure for model accuracy, the degree of accuracy of the model of the target system can be accounted for to increase the usefulness of model-based reasoning in both monitoring and evaluation of sensor placements.

Acknowledgements

This work was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

[Bose and Rajamoney 91] P. Bose and S. Rajamoney, "An Approach Based on First Principles For The Design of Continuous Devices," *Proceedings of the 1991 Workshop on Model-based Reasoning*, Anaheim, CA, 1991.

[Chien et al. 91a] S. A. Chien, R. J. Doyle, and L. S. Homem de Mello, "A Model-based Reasoning Approach to Sensor Placement," *Proceedings of the 1991 Workshop on Model-based Reasoning, Anaheim*, CA, 1991.

[Chien et al. 91b] S. A. Chien, R. J. Doyle, and N. Rouquette, "A Model-based Reasoning Approach to Sensor Placement for Diagnosability," *Proceedings of the Second International Workshop on the Principles of Diagnosis, Milan, Italy*, 1991.

[Doyle et al. 89] R. J .Doyle, S. M. Sellers, and D. J. Atkinson, "A Focused Context-sensitive Approach to Monitoring," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, 1989.

[Doyle et al. 91] R. J. Doyle, U. M. Fayyad, D. Berleant, L. K. Charest, L. S. Homem de Mello, H.J. Porta, and M.D. Wiesmeyer, "Sensor Selection in Complex Systems Monitoring Using Information Quantification and Causal Reasoning," in Recent Advances in Qualitative Physics, B. Faltings and P. Struss (eds.), MIT Press, 1991.

[Dvorak and Kuipers 89] D. Dvorak and B. Kuipers, "Model-based Monitoring of Dynamic Systems," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Detroit*, MI, 1989.

[Davis and Hamscher 88] R. Davis and W. C. Hamscher, "Model-based Reasoning: Troubleshooting," In H. E. Shrobe, editor, Exploring Artificial Intelligence: Survey Talks from the National Conference on Artificial Intelligence, Morgan Kaufman, San Mateo, CA 1988.

[Scarl et al., 88] E. A. Scarl , J. R. Jameison, and E. New, "Deriving Fault Location and Control from a Functional Model," *Proceedings of the Third IEEE Symposium on Intelligent Control*, Arlington, VA, 1988.

An Architecture for the Development of Real-Time Fault Diagnosis Systems Using Model-Based Reasoning

Gardiner A. Hall, James Schuetzle,
David LaVallee, and Uday Gupta

Loral AeroSys
7375 Executive Place, Suite 101
Seabrook, Maryland 20706
(301) 805-0300

Abstract

This paper presents an architecture for implementing real-time telemetry-based diagnostic systems using model-based reasoning. First, we describe Paragon, a knowledge acquisition tool for offline entry and validation of physical system models. Paragon provides domain experts with a structured editing capability to capture the physical component's structure, behavior, and causal relationships. We next describe the architecture of the run-time diagnostic system. The diagnostic system, written entirely in Ada, uses the behavioral model developed offline by Paragon to simulate expected component states as reflected in the telemetry stream. The diagnostic algorithm traces causal relationships contained within the model to isolate system faults. Since the diagnostic process relies exclusively on the behavioral model and is implemented without the use of heuristic rules, it can be used to isolate unpredicted faults in a wide variety of systems. Finally, we discuss the implementation of a prototype system constructed using this technique for diagnosing faults in a science instrument. The prototype demonstrates the use of model-based reasoning to develop maintainable systems with greater diagnostic capabilities at a lower cost.

I. Introduction

Diagnosing spacecraft faults is a difficult, error-prone, and time-consuming activity. Spacecraft diagnosis is performed by an operations team composed of a large contingent of highly trained people. These people monitor a satellite telemetry stream containing hundreds of system data points. When an anomaly is detected, the operations team analyzes this data with respect to archived historical telemetry data and detailed spacecraft design information. Analyzing such large quantities of data and developing a hypothesis explaining the data is an extremely challenging task. It is not uncommon for satellite anomaly investigations to take several days.

The already difficult chore of satellite fault diagnosis will be even more demanding in the future. Satellites and their instruments will become more sophisticated and complex, raising the complexity of the fault analysis process. Along with increased complexity, future missions are expected to last longer. A mission life measured in terms of decades rather than years, introduces challenges in maintaining the operations team skill level. The desire to support interactive science operations conducted by people external to the control center will further complicate fault diagnosis activities. The operations crew's ability to maintain a current accurate assessment of the spacecraft's state will be

taxed as more people manipulate the spacecraft and its instruments. Due to these increased complexities, the corresponding control centers are apt to be more costly to build, maintain, and operate.

The application of artificial intelligence techniques promises to help alleviate these problems by increasing the level of automation in spacecraft operations. Specifically, improving the automation level of a control center may result in realizing the following benefits:

a. reducing the risk of catastrophic mission failures

b. reducing the cost of control center operations

c. increased spacecraft and instrument utilization

d. increased retention of key operator's skills

e. an ability to "scale up" control centers to handle more complex spacecraft, more spacecraft and instrument activities, and more users without a proportional increase in cost

This paper describes a system that improves the level of automation in a control center by automating a control center's fault detection and isolation activities.

Background

Our approach to providing automated fault diagnosis tools that quickly and accurately find and solve problems is centered on three basic premises. First is the belief that knowledge base construction and maintenance activities are most appropriately performed by domain experts. Second, a fundamental feature of our expert systems is the separation of problem solving from knowledge acquisition. Third, the tools we build reflect the notion that solving different problems requires different problem-solving techniques. The rationale for this design philosophy is documented in [JAW-87]. Figure 1 illustrates the architecture derived from these design principles.

Our first tool, a rule-based expert system, the Ford Lisp Ada Connection (FLAC) described in [JAW-88], includes an offline knowledge acquisition component and an online inference engine. The offline component is an intuitive graphical editing tool that is used directly by the domain expert. It does not require knowledge of AI or expert systems and is easily learned by the domain expert. The rule base is developed as a graph of nodes symbolically depicted as *and/or gates*, as typically seen in CAD systems for integrated circuit design. Once the expert is satisfied with the rule base it is downloaded to the online system. The rule base is loaded into data structures at run time for use by the embedded Ada inference engine.

FLAC successfully demonstrated the feasibility of real-time expert systems. However, the limitations of production rule systems soon became apparent. Fundamental to these systems is the requirement to enumerate explicitly all possible faults. Intuitively, as the complexity of the system increases, it becomes increasingly difficult to predict accurately every possible fault scenario. Another deficiency in the rule based approach is the inability to gracefully solve problems that change over time. One key requirement for a diagnostic system is the capability to reason about temporal and control relationships between attributes of the target system. Developing a rule base that captures and implements rules describing temporal and control relationships is exceedingly difficult and error-prone.

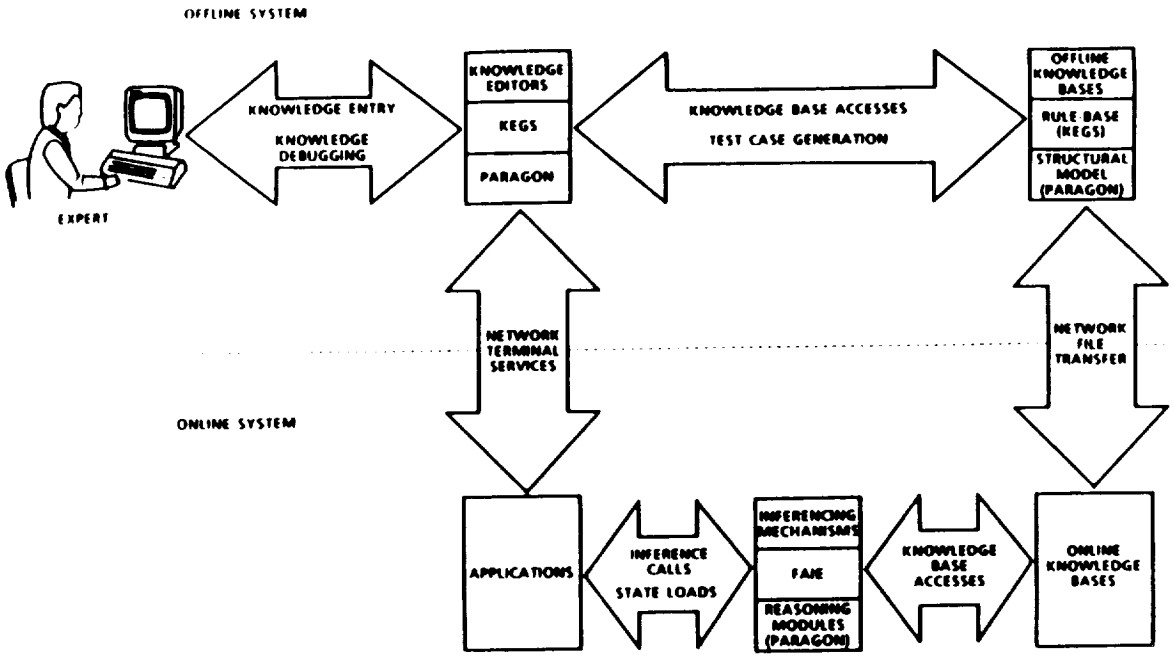


Figure 1. An Architecture for Knowledge-Based Systems

Issues associated with maintaining a rule base large enough to describe a spacecraft also became apparent. Because of the unstructured nature of rule bases, maintenance is difficult when adding or modifying rules. The unstructured nature of rule bases also leads to a formidable verification and validation task. Additionally, as rule bases become larger, maintaining consistency between rules becomes increasingly difficult. Maintaining rule base integrity requires the addition of more rules and routines dedicated to the consistency checking function.

To overcome these difficulties, we began an investigation into a model-based reasoning approach to real-time fault diagnosis. The model-based reasoning approach has promising features relevant to control center fault diagnosis activities. For instance:

a. Model-based systems reason from deeper principles. Model-based systems know the internal processes of a machine and can determine the machinery's state from observed values. In a rule-based system, relationships defining each observation and the machine's state must exist.

b. Model-based systems can reason about a system as it changes over time. Model-based reasoning systems have this capability because events and conditions can be represented by mathematical functions that are close approximations of actual conditions.

Like FLAC, our model-based reasoning system contains an offline graphical component for easy entry of knowledge, and an online embedded diagnostic component. The offline component, originally implemented by Loral's Space and Range Systems division, is a model-building tool called Paragon. Paragon is used to build a structural and functional model of the system to be monitored. The model is exported as a

file to be loaded at run time by the online diagnostic component. The diagnostic system, developed by Loral AeroSys, uses the behavioral model to predict expected states for the system and compares them to actual states as reflected in the telemetry stream. The diagnostic algorithm traces causal relationships described in the model to isolate system faults. The diagnostic system is implemented in Ada and is capable of real-time performance on conventional processors.

The remainder of this paper discusses our experiences with the model-based reasoning approach in more detail. Section II describes the architecture of the system. The implementation of the prototype is covered in Section III, and our results and conclusions are presented in Sections IV and V respectively.

II. Model-based Reasoning System Architecture

In our prototype system there is an offline component for creating and verifying knowledge bases, and an online component for the diagnostic software. This design reflects the architecture of many current control centers (e.g., MSOCC, Space Telescope). The offline systems define telemetry and command databases, while the online systems use these databases for interpreting spacecraft telemetry and building spacecraft commands.

Offline Knowledge Acquisition System

The Paragon knowledge acquisition tool provides a method to construct a detailed structural and functional model of a problem domain. The model is specified in terms of objects, object behaviors, and relationships between objects. These different views of a model can be thought of as defining *conceptual* and *relational* entities. Conceptual entities define concepts existing in the problem domain and are composed of *dynamic* and *static* aspects. Dynamic aspects describe an object's relationships to other objects and how that object may be manipulated. Static aspects describe the object's attributes and how these attributes relate to other concepts. Relational entities describe relationships between two concepts. Each relationship within the model has a specific and well-defined behavior. Figure 2, a screen dump from a Paragon session, provides an example model definition.

Concepts in the Paragon system are either relations, classes, or instances. The Paragon system supports inheritance in the form of **class ⇨ subclass ⇨ instances.** This classification scheme is a strict hierarchy; an instance may have at most one defining class. Using this scheme, a semantic network is constructed representing the real-world system. The frames composing the network are the defined instances. The slots of the frame hold the object's attribute values. Relation objects link the frames to complete the network.

Figure 3 provides an example of applying concepts, relations, and dynamic and static aspects to a physical object, a thermal switch. Two relationships, *temperature* and *current*, affect the concept *thermal switch*. The switch also contains the local attributes *switching temperature* and *output*. The internal process of the thermal switch provides for two possible states: *ON* and *OFF*. The dynamic aspects of the concept of the thermal switch are represented by the links labeled *Temperature >= SW* and *Temperature < SW*. These represent the possible transition conditions between the ON and OFF states. For example, if the incoming temperature value is less than the local attribute switching temperature, control is passed to the ON state. The static aspects of the thermal switch concept are described by the event equations labeled *Output = 0* and *Output = Current*.

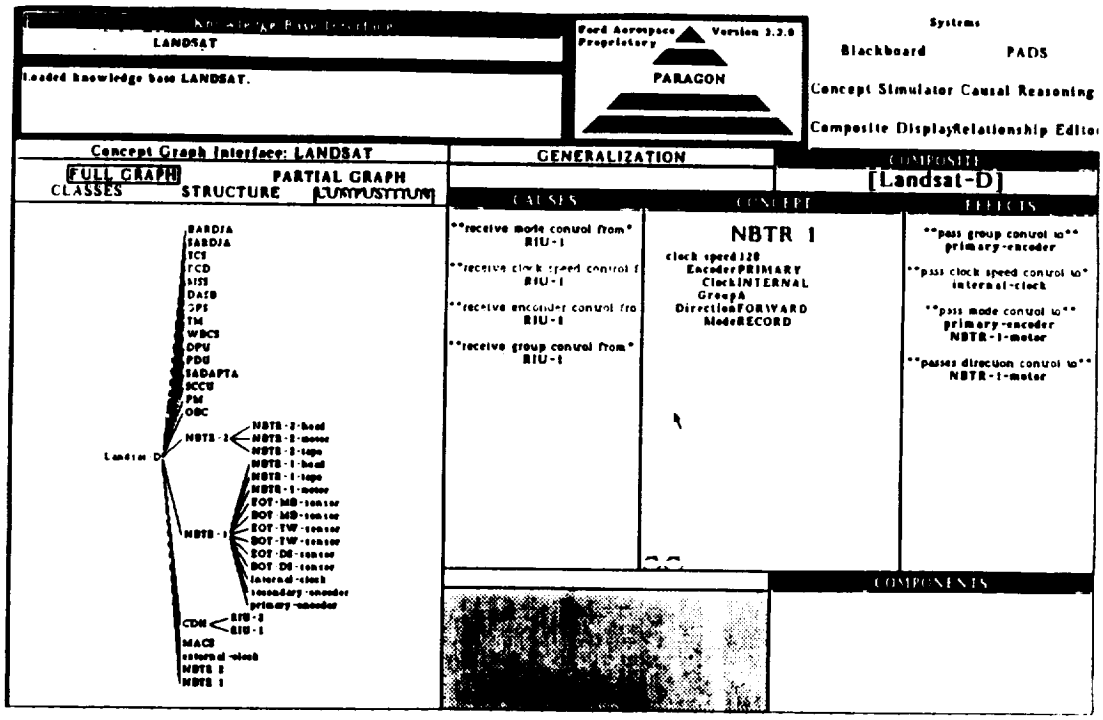


Figure 2. Sample Paragon Knowledge Base

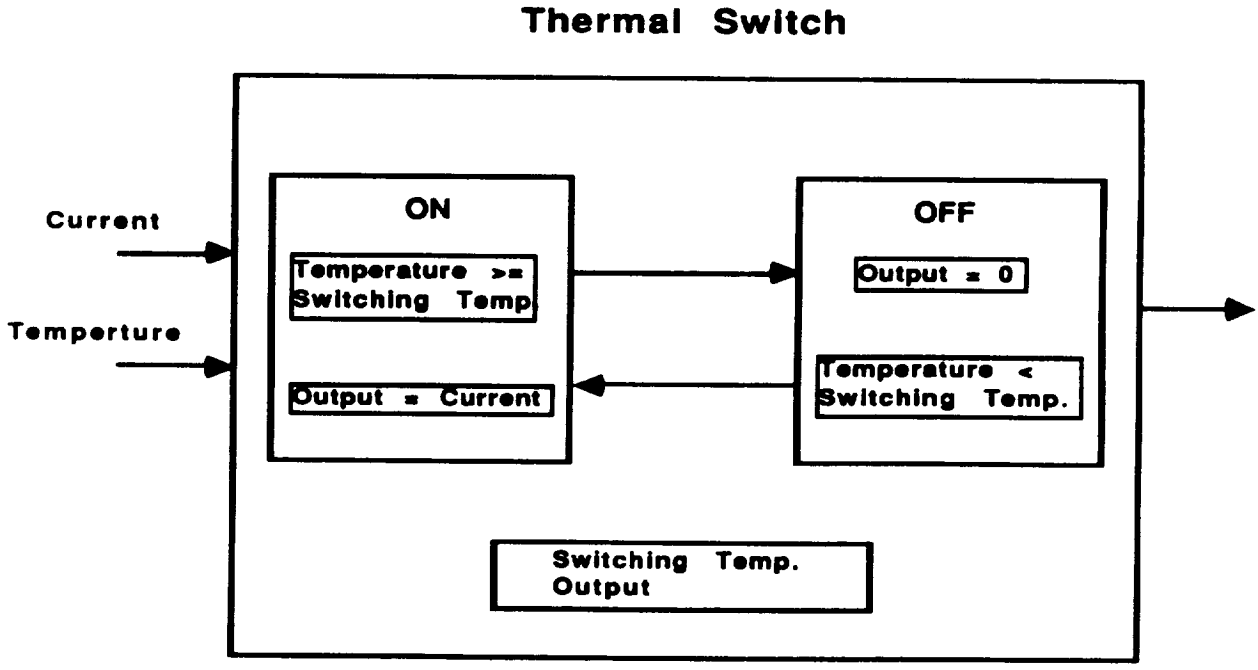


Figure 3. Thermal Switch Class Example

Using these ideas, a model of the physical system is built by recursively defining and instantiating frames and relationships. For a spacecraft, the objects in the system are the onboard components. Each object's attributes are the parameters contained in the spacecraft's telemetry stream. The design and functional information for each object is captured by defining the object's possible states, state transitions, and behavior when in a particular state. An object's attributes may be affected by other local attributes, itself, or attributes of another object. The process of defining objects and relationships continues until a model of satisfactory fidelity is achieved. Included with Paragon are tools for inspecting the classes, objects, and relationships within the system. Also included is a simulation capability for validating the model's correctness.

Online Diagnostic System

The real-time component of the diagnostic system is the Model-Based Reasoning (MBR) module. The primary function of the MBR module is to detect and diagnose electromechanical or other system faults in real time. The diagnostic system is composed of simulation, monitoring, and causal analysis subsystems. The simulation subsystem uses the Paragon-developed knowledge base to generate expected values for each telemetry (attribute) point. The monitoring subsystem synchronizes the simulation with actual time, and performs expected versus observed value comparisons. A mismatch between these two values triggers the causal-analysis subsystem. The causal-analysis subsystem develops hypotheses explaining the observed behavior by examining each faulty component's relationships. These three subsystems work in unison to perform fault detection and diagnosis.

The simulation subsystem uses the information contained in the Paragon model to continuously update the target system's expected state. Specifically, the simulation cycles through all the objects in the system, evaluating each object's state transition criteria for the current state. Once the current state is determined, its attributes are modified to reflect that state. The frequency of the simulation's cycle is the real-time rate. Modifications to the expected state can be effected through external commands, scheduled activities, or the model's internal processes. Maintaining this model provides a reference point for evaluating the spacecraft's health.

The monitoring subsystem is responsible for fault detection. The monitoring process compares time-synchronized, simulation-generated, expected values with actual system-measured values (telemetry in the case of a space system) at predefined time intervals (cycle). A component is considered to be abnormal when these two values disagree. These abnormal components, along with their attributes, actual and expected attribute values, and fault-detection cycle identifiers, are posted to a blackboard structure, called the Abnormal-Components-Blackboard. The Abnormal-Components-Blackboard is inspected to determine if the detected abnormal component exists on the blackboard. If the component does exist on the blackboard, its fault-detection time-cycle identifier is updated with the old fault-detection time-cycle number before it is posted to the blackboard. Whenever abnormal components are detected, further analysis is performed by the causal analysis subsystem to isolate the exact cause(s) of the fault(s) from the abnormal components list.

The causal analysis of suspected abnormal components relies on functional and design information provided by the Paragon model. The basic fault-diagnosis strategy for the causal analysis is:

a. The list of suspected components is read from the Abnormal-Components-Blackboard. A node corresponding to each suspected component is created. These nodes are referred to as Fault Mechanism Nodes (FMN) and are maintained in a list structure.

b. Design and causal link information is obtained for each faulty component.

c. During this step, the causal-effect pointers of the FMNs are assigned. Three types of pointer are set: In-link, Out-link, and Next pointers. In-links point to FMNs whose components affect the attribute(s) of the current FMN. FMN Out-links point to FMN(s) whose component(s) attribute(s) can be affected by the current FMN. The Next pointer simply points to the next FMN. Setting In-link, Out-link, and Next pointers transforms the FMN list into a graph, referred to as a Fault Mechanism Graph (FMG). Figure 4 shows a FMG. Each block contains the component name, In-link, Out-link, and Next FMN pointers. As shown in Figure 4, the component Power Supply 1 has a null In-link pointer indicating that it is not affected by any other FMN. The Out-link pointer of Power Supply 1 points to the node Instrument Power. This indicates that Power Supply 1 causes Instrument Power to be abnormal. Instrument Power's In-link pointer indicates that Instrument Power is affected by Power Supply 1. The component VNIR FPA has a null Out-link pointer indicating that it does not affect other FMNs. These interpretations can be similarly applied to the other nodes of the graph.

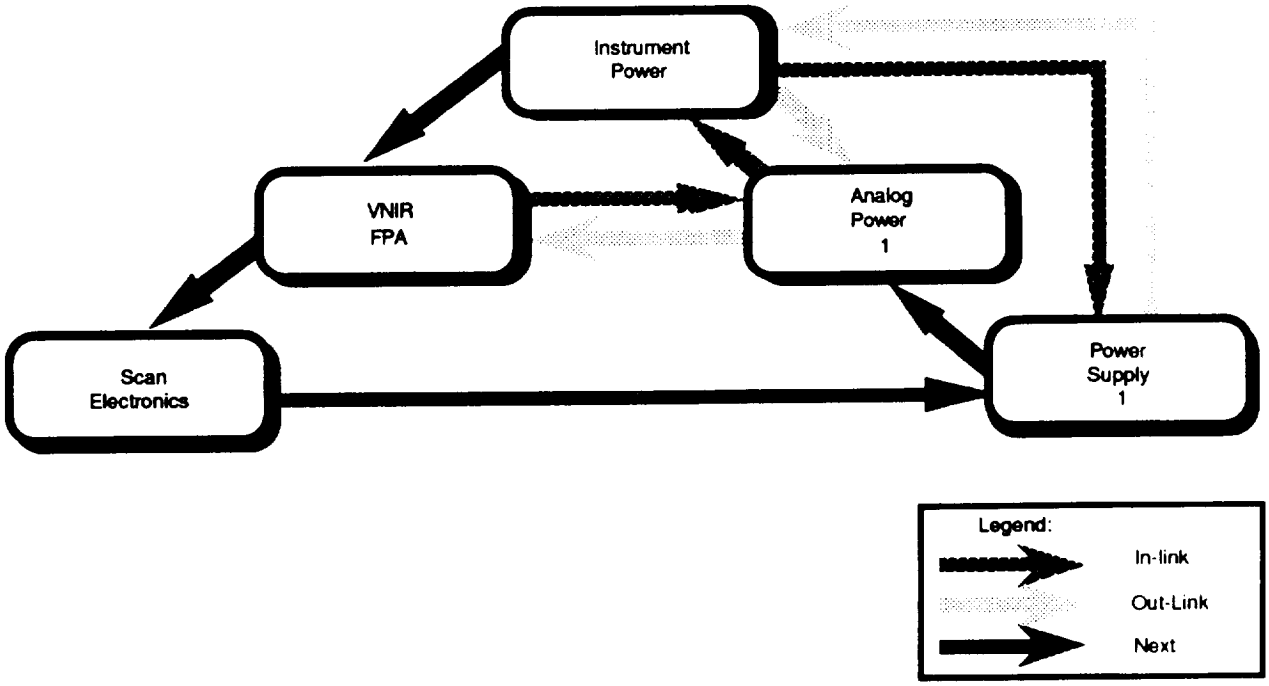


Figure 4. MODIS Fault Mechanism Graph

d. The In-link and Out-link pointers of each node of the FMG are examined. Components with null In-link pointers are considered to be fault sources. Fault-propagation paths are computed by iteratively selecting those FMN's with null In-link pointers, tracing the node's Out-link pointer to the affected FMN and tracing the affected FMN Out-link pointer to other affected FMNs until the current Out-link pointer is null. These paths explain the order in which components became abnormal.

Steps a through d are repeated when a fault-detection cycle detects an abnormal component, or a previously detected abnormal component is found to have returned to a normal state.

III. Prototype Implementation

We demonstrate our model-based approach for real-time fault detection and diagnosis in a testbed environment. The testbed is a complete command and control environment for the Moderate Resolution Imaging Spectrometer (MODIS), a future Earth Observing System (EOS) instrument. Our prototype runs on a VaxStation 3100 and is implemented in the Ada programming language. The MODIS model was developed using the PARAGON tool on a Symbolics 3640 and downloaded to the Vax workstation.

The model, based on a proposed design for the MODIS instrument, took three months to implement using Paragon. The MODIS model consists of over 50 component classes, 80 components, and 11 types of functional relationship. In addition, the model is capable of responding to 96 different instrument commands, and transmits 132 different telemetry points. The model includes definitions for all normal instrument states, state transitions, and internal attribute update equations for all components.

The testbed contains three processors to provide a high-fidelity environment for evaluating control center automation techniques. The architecture of the testbed is shown in Figure 5.

The Symbolics is the offline processor, used for creating knowledge bases. One of the VaxStations is dedicated to control center functions. In addition to fault diagnosis, there is software for:

a. Receiving and decommutating a 2 Kbs stream of packetized telemetry

b. Processing and transmitting instrument commands

c. Displaying graphically instrument telemetry data

The other VaxStation, the telemetry source, executes simulation software generating instrument telemetry. The simulator has the capability to:

a. Modify the value of any object's attributes

b. Update the current state of an object

c. View any object's attribute values

d. Control the length of simulation cycle time (useful for debugging)

e. Accept and process instrument commands sent from the ground

f. Packetize and transmit telemetry.

Telemetry and commands are exchanged between these two processors by way of Ethernet using the TCP/IP protocol.

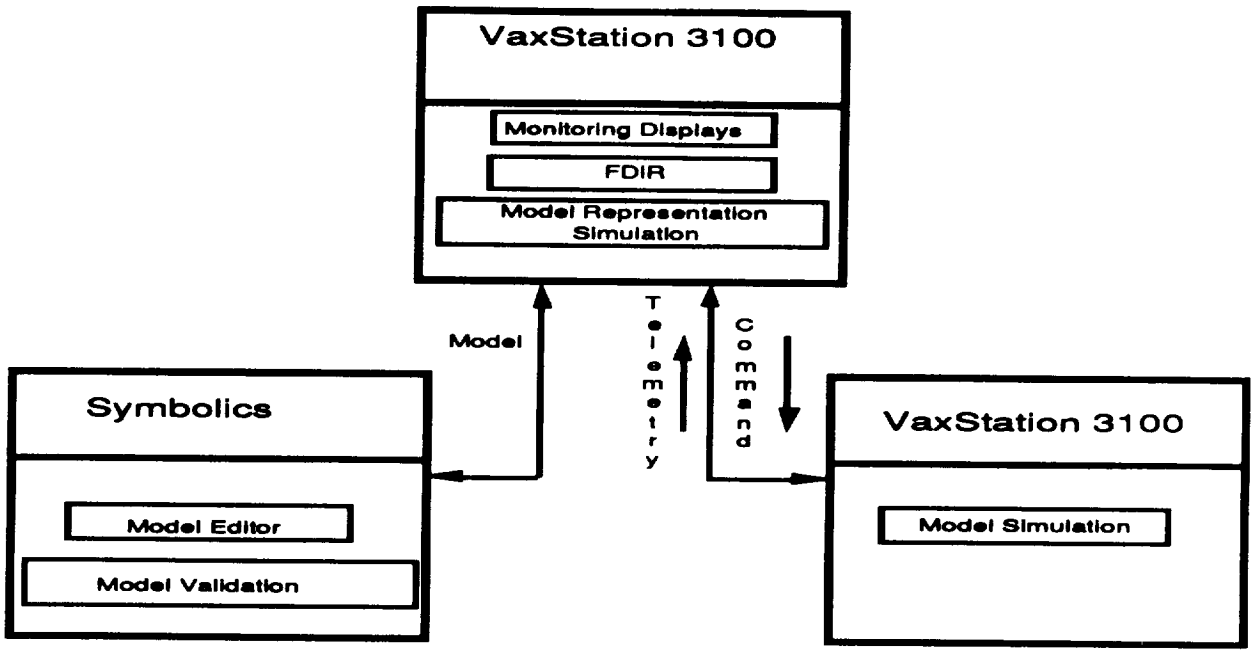


Figure 5. MODIS Control Center Testbed

IV. Results

Using the testbed, the prototype MBR system has been tested under several different fault scenarios. These fault scenarios were developed on the basis of Loral's spacecraft operations experience. The types of fault scenarios tested were:

a. Components that commonly fail during mission (e.g., a sticky relay)

b. Rare or infrequent component faults (e.g., a failed door drive motor)

c. Multiple simultaneous fault scenarios (e.g., a failed heater and a faulty relay).

We also tested the case where two components affecting a common component fail. In this scenario, our prototype identified both components as failed. In all fault scenarios tested, the MBR system accurately detects and isolates the source of fault. Importantly, these fault scenarios were designed after the implementation of the model and fault diagnostic software. In no case were component specific rules describing fault conditions or causes implemented.

V. Conclusions

These preliminary results suggest that model-based reasoning is a viable method for automating spacecraft fault detection and diagnosis activities. On the basis of this research several advantages of the technique are apparent:

a. A model-based system is capable of detecting and diagnosing unpredicted, non-intuitive faults in a continuous, dynamic system in real-time

b. The knowledge acquisition process is simplified

c. The maintenance of the knowledge base is simplified

d. This technique can be leveraged with other control center and spacecraft implementation efforts.

This system has the capability for detecting and diagnosing unpredicted faults. The test cases that have been devised emphasize this point. The design of the fault scenarios used for test purposes was based upon operational requirements rather than diagnostic system capabilities. During demonstrations of the prototype, faults are generated "on the fly" by having members of the audience select the component to be faulted.

Knowledge acquisition activities for implementing the system are reduced. Building the knowledge base for a model-based reasoning system only requires the ability to describe a correctly operating system. Since there is no need to enumerate all possible behaviors, the amount of time required for constructing knowledge bases is reduced. Verification of the knowledge base is easier. Using the physical system as a reference point allows a simple comparison demonstrating the model's accuracy. One of the advantages of representing a physical system as a network of objects is that this presentation lends itself to a graphical representation. A graphical representation is advantageous because it allows the knowledge-base builder to view components from different perspectives.

An important advantage a model-based has over rule-based systems is that knowledge-base maintenance is an easier task. The object orientation of the model simplifies knowledge base maintenance. As each object's interfaces with other objects in the system are clearly defined, modifications can be localized to the object, reducing the potential for harmful side effects. The object-oriented approach also provides for potential knowledge-base reuse. For example, libraries containing generalized reconfigurable objects can be built.

The single most important advantage of a model-based system may be that it is complementary to current control center designs. Calculating the expected state for each on-board component provides a mechanism for dynamically updating telemetry limits and alarm values. Another key point is that most projects construct simulators for ground system verification and training. The model-based technique for fault diagnosis provides a method for leveraging these simulators into day-to-day operations.

References

[JAW87] Jaworski, A., LaVallee, D., Zoch. *A Lisp-Ada Connection,* Proceedings of the 1987 Goddard Conference on Space Applications of Artificial Intelligence and Robotics, (1987).

[JAW88] Jaworski, A., Thompson, J. *Automated Satellite Control in Ada,* Proceedings of the 1988 Goddard Conference on Space Applications of Artificial Intelligence and Robotics, (1988).

Bibliography

Items not specifically referenced in the text.

Ferguson, J.C., Siemens, R.W., Wagner, R.E.*STAR-PLAN: A Satellite Anomaly Resolution and Planning System,* proceedings of AAAI Workshop on Coupling Symbolic and Numerical Computing in Expert Systems, (Aug. 1985)..

Fulton, S., Pepe, C. *An Introduction to Model-Based Reasoning*, (January 1990), AI Expert (pp. 48 - 55).

The Achievement of Spacecraft Autonomy Through The Thematic Application of Multiple Cooperating Intelligent Agents

Philip J. Rossomando
General Electric Military and Data Systems Organization
(215) 531–5087

King of Prussia, Pennsylvania

KEYWORDS: Autonomy, Automation, Blackboard, Real–time Diagnosis, Expert Systems, Theme, Role Playing

1.0 ABSTRACT

This paper presents a description of UNICORN, a prototype system developed at General Electric for the purpose of investigating Artificial Intelligence (AI) concepts supporting spacecraft autonomy. With this objective, UNICORN employs thematic reasoning, of the type first described by Rodger Schank of Northwestern University, to allow the context–sensitive control of multiple intelligent agents within a blackboard–based environment (Schank, 1977). In its domain of application, UNICORN demonstrates the ability to reason teleologically with focused knowledge. Also presented within the following sections are some of the lessons learned as a result of this effort. These lessons apply to any effort wherein system level autonomy is the objective.

2.0 INTRODUCTION

A space–based system is composed of many subsystems whose associated performance can each contribute significantly to the success or failure of a mission. Each of these subsystems has its own changing needs and possibly conflicting requirements, which must be reconciled to maintain overall spacecraft health and operability. To address these issues, the solution space can be partitioned into multiple abstraction levels along both physical and functional boundaries. This partitive approach to spacecraft autonomy can be complemented through the context sensitive application of both conventional and advanced AI techniques within a hierarchical distributed control structure of the type currently found mostly within research institutions. The selective application of independent intelligent agents brings significantly more applicable knowledge to bear on a problem than is possible through the utilization of either conventional expert system or procedural methodologies alone.

For the past several years, GE in King of Prussia has been exploring AI problem solving paradigms that it feels will ultimately lend themselves to autonomous spacecraft operation. The first of these, rule/experiential and case–based reasoning, while allowing diagnostic knowledge to be expressed explicitly in the form of if–then rules and structured objects, falls prey to boundary conditions and is limited by the need for the *a priori* definition of faulted models and past cases. On the other hand, model directed reasoners of the type originally proposed by Johan de Kleer and Randal Davis are extremely CPU intensive ((deKleer, 1987), (Davis, 1985)). This fact limits their applicability within real–time environments of the type within which an autonomous spacecraft is expected to operate. To achieve autonomous operation, what seems necessary is a problem solving paradigm that allows the combining of the benefits of these approaches while at the same time minimizing their inherent weakness.

To attain the hybrid operation alluded to requires the ability to choose between knowledge sources employing both deep and shallow reasoning, based upon the current operational context of the space platform. This context need not necessarily be derived from the physical environment alone, but may arise from the goals and expectations identified to the spacecraft before and during operation. The blackboard control structure first introduced within the HEARSAY II environment appears to allow for this cooperative application of diverse knowledge sources (Erman, 1974). The twist however, introduced at GE, is not to opportunistically apply knowledge blindly but rather to do so in a focused manner that takes into account the spacecraft's context and the competing goals and demands of each of the subsystems of which the spacecraft is composed. This capability requires the utilization of intelligent agenda schedulers that understand these underlying needs while at the same time possessing a teleological comprehension of mission objectives within the constraints imposed by their assigned roles. This fact is significant because the goal of UNICORN is not just automation of space system functionality, but rather, is true autonomy.

While both autonomy and automation share attributes, and while both imply a reduction in human physical work load, autonomy also *implies* an *understanding of purpose*. This fact can be utilized to achieve a significantly greater reduction and/or elimination of the requirement for human cognitive activities related to spacecraft administration. Only through a deep understanding of mission objectives provided through the utilization of AI technologies supportive of autonomy, such as those herein documented, can we expect to produce self–directed spacecraft. By understanding why it was created, the autonomous system can be placed in a better position to prioritize its activities, utilize scarce resources, and generate expectations so as to achieve complex objectives. The question thus becomes how to develop a system wherein an understanding of both mission and ability can be utilized to guide performance in a highly unpredictable and constantly changing environment.

2.1 Background

The following sections provide the background needed in order to comprehend the difficulty of the spacecraft autonomy problem and why Thematic Reasoning within a blackboard–based control structure was chosen to address this issue.

2.1.1 Development History

GE began its investigation of autonomy by attempting to identify those spacecraft–related tasks where autonomy seemed most applicable. As indicated in Figure 1, spacecraft functionality can be divided into three areas:

1. Mission Management
2. Health & Maintenance
3. Payload Operations

Mission management relates to those tasks necessary to ensure that the payload can perform its assigned task. Such activities as on–board orbit maintenance and resource management fall under this heading. Under normal circumstances, an unmanned spacecraft's *Prime Directive* is to achieve as many payload mission objectives as possible. It is the duty of the Mission Manager to see to it that the overall spacecraft functions smoothly in addressing this directive. A payload is the package carried by the BUS that performs the task(s) for which the spacecraft was constructed.[1] Payload Operations determines what and when payload related activities are to be performed and makes demands of spacecraft resources based on these objectives.

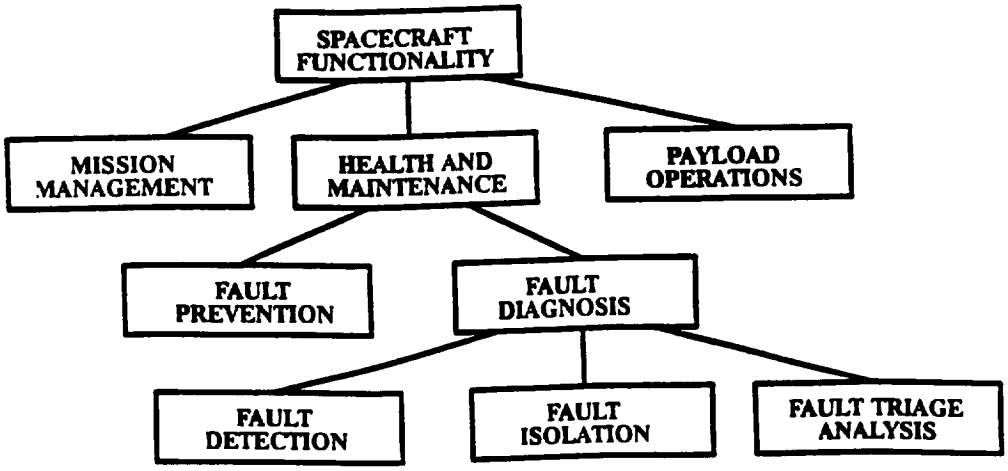


Figure 1. A Taxonomy Of Spacecraft Functionality

1. We use the terms BUS and spacecraft interchangeable within this paper.

Assuming an active payload, the demands placed on the BUS by the Mission Manager as a result of payload objectives can be expected to change with time. These demands are addressed by spacecraft BUS dedicated subsystems such as:

1. Power System
2. TT&C System
3. Thermal System
4. ACS

For example, the Electrical Power and Distribution System(EPDS) is expected to provide electrical power to the payload and the BUS's subsystems, while the Attitude Control System (ACS) is expected to maintain the spacecraft's stability. The Health and Maintenance function is responsible for ensuring that when resource/functionality demands are made of these subsystems they can be adequately addressed. This is done by:

1. Monitoring BUS related–subsystem operation relative to contextual expectations, thereby detecting possible fault–related symptoms.

2. Identifying trends that may imply future failures for which early corrective action can be taken.

3. Performing fault isolation and triage analysis.[2]

Because of a variety of factors, these are not easy tasks to make autonomous.

One significant reason for this difficulty is that the BUS subsystems are complex devices with many interacting subsystems of their own. Each of these is in turn composed of many parts whose behavior is critical to proper spacecraft operation. Faults within any of these components can cause adverse symptomatic behavior in the other components both within and external to the faulted subsystem. In addition, the behavior of spacecraft components can change without the presence of a fault anywhere within the spacecraft. The cause of an anomalous component behavior may be due to some change in the external spacecraft context. As indicated in Figure 2, in the course of its lifetime, an earth–orbiting satellite will change position relative to the earth, the moon, and the sun many times. Depending on that position, the behavior of critical satellite components can be expected to vary.

2. The UNICORN system performs triage analysis after it isolates the cause of a failure. In the case of a failed sensor, the identifying agent may choose to remove it from the scan list. If a backup system exists or a work around is possible, this may be provided as a recommended correction. On the other hand, there may exist no possible corrective action, and the spacecraft will then attempt to make do as best it can.

The electrical output of a solar array for example, can drastically change when the satellite is eclipsed by the earth or moon, without even a fault in that component. In the process of investigating spacecraft contexts, it was determined that these need not be just physical but may also be temporal (Allen, 1984). Component aging can also have a significant effect on behavior expectations. While these context changes can be predicted, others cannot and may appear as something other than what they actually are. For example, a nuclear burst exhibits many of the same characteristics as a solar flare. Another significant factor that further complicates the Health and Maintenance function is that even after a fault is corrected, spacecraft behavior does not instantly return to normal. During this period, if constraints are not relaxed, symptomatic behavior can be expected to continue even in the absence of a new fault.

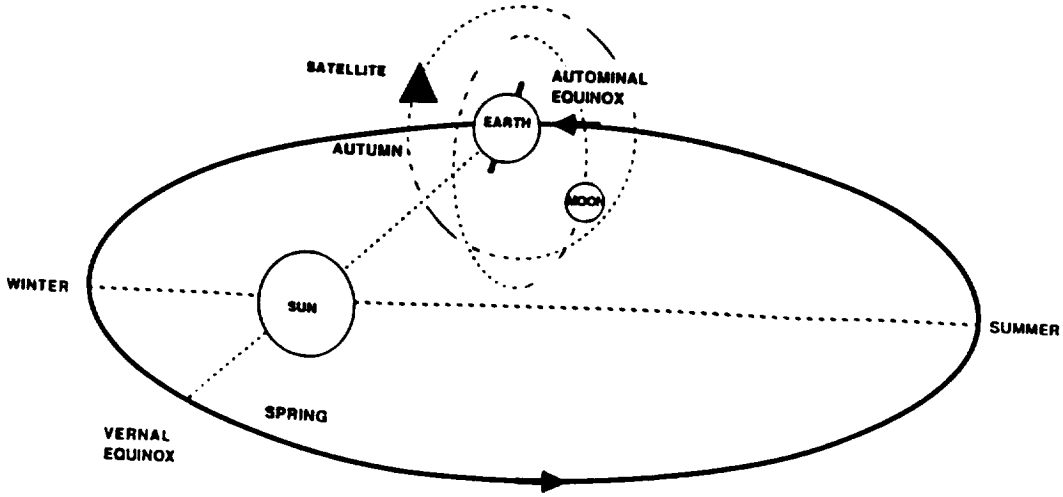


Figure 2. A Spacecraft's Context Is Dynamic

In 1986, in an attempt to automate the spacecraft Health and Maintenance function, GE utilized KEE on a Symbolics 3640 computer to develop a rule–based expert system, applying W. Clancy's classification strategy for performing diagnosis on a portion of the DSCS ACS ((Clancey, 1984), (Bell, 1986)).[3] While successful in its application domain, it soon became evident that enough rules could never be identified so as to ensure the diagnostic autonomy of an ACS let alone that of an entire spacecraft. In addition, it was discovered that if a fault exhibited symptoms slightly different from those expected, the rule–based system could not identify its cause.[4] Thus, the utilization of model–directed reasoning with constraint propagation and constraint relaxation was decided upon. This seemed to be an ideal approach as GE has a wealth of first–principle satellite knowledge readily available; by reasoning from first principles, faulted models are no longer necessary. Model–directed reasoning also seemed appropriate, as it appeared the only way to explicitly represent the peripheral paths of causal interaction introduced by the environment and adjacent, but not connected,

3. KEE is a trademark of IntelliCorp.
4. Even the loss of one sensor can completely blind a rule–based diagnostic expert system and cause it to arrive at erroneous conclusions.

subsystems ((Abbott, 1990), (Davis, 1985)). To investigate the accuracy of these predictions, in 1987 a model directed system was developed, also in KEE, to diagnose faults within an ACS. This system worked well but brought to light problems that originally were not apparent (Rossomando, 1988). For example, back propagation is not possible when the component's transfer function has no inverse, thus making other strategies necessary if fault isolation is to continue. However, because of the basic success of this effort, it was decided to tackle the system illustrated Figure 3.

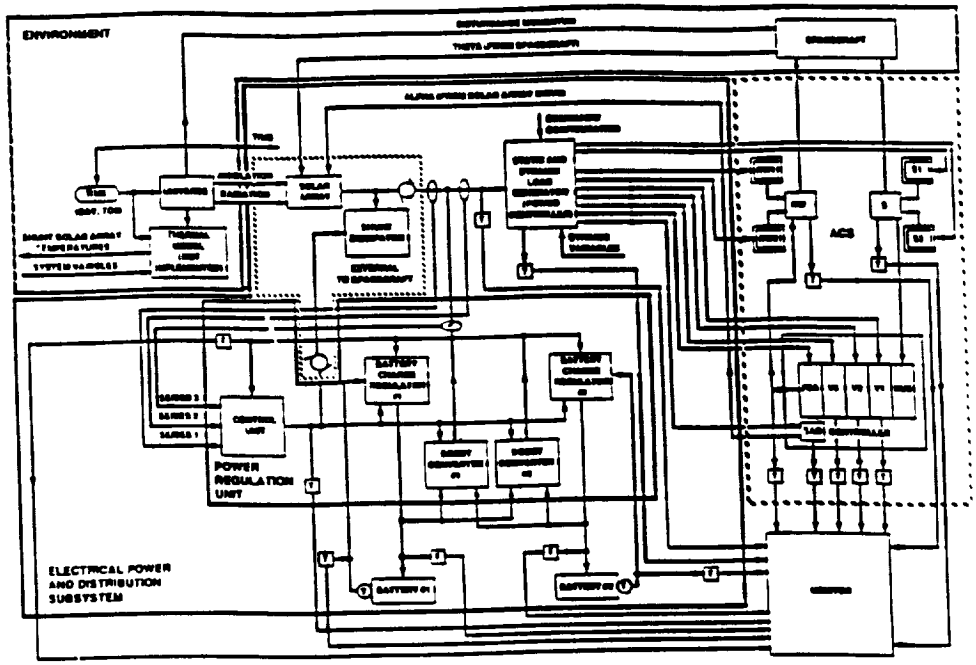


Figure 3. The UNICORN Spacecraft and Environment Model

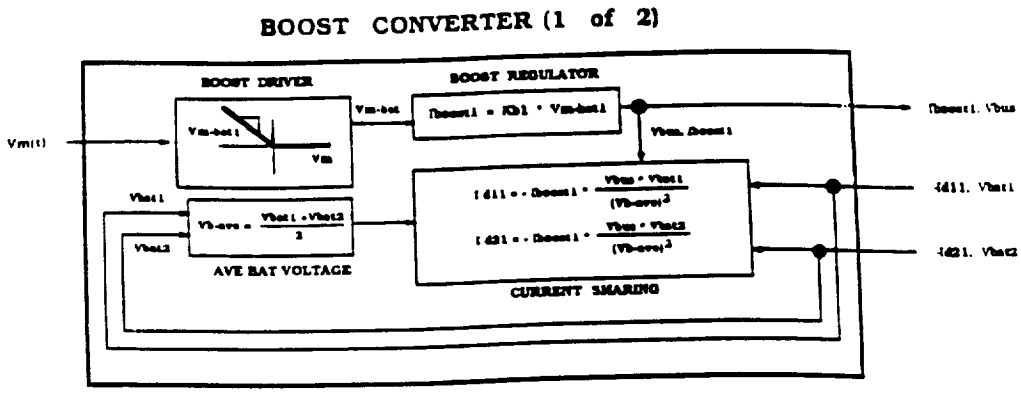


Figure 4. The Boost Converter

In the process of isolating faults within this more complex spacecraft, it was observed that constraint

propagation can be very CPU intensive.[5] In an effort to address this problem, the spacecraft was divided into three separate but related causal models:

1. The Power System.
2. The ACS.
3. The External Environment.

In addition, it was observed that certain fault types have signatures that make them ideal fault model candidates. As a result, it was further decided that some combination of deep and shallow reasoning would be most appropriate in addressing real–time spacecraft Health and Maintenance issues. The question now was how to best combine these techniques in an environment that would allow reasoning across subsystem boundaries while also supporting detection, resolution, and mission demands not associated with fault isolation. Thus was born the UNICORN effort. Before describing that effort, however, the following sections introduce some of the non–diagnostic technologies on which UNICORN is based.[6]

2.1.2 The Blackboard Control Structure

A blackboard is much more than a shared global area.[7] It is that plus a whole collection of intelligent agents/actors and concepts that must work together to solve a complex problem. As indicated in Figure 5, a blackboard can be divided into multiple abstraction levels called panels. Each panel can be made to correspond to a different degree of solution abstraction. Associated with each panel are a number of *knowledge sources* whose purpose is to do certain specific sub–tasks within a much larger problem space. Each knowledge source in turn is an independent, self directed, intelligent agent composed of a number of condition and action parts. Knowledge source communication is strictly through the creation of *blackboard events*.[8] Some knowledge sources, such as fault isolators, symptom detectors, and triage analysts, are domain specific while others have generally more domain independent responsibilities. Two agents/actors in particular are worthy of note. These *control knowledge sources* are:

---

5. Within the model illustrated in Figure 3, as necessary, each identified component was itself modeled. For example, as illustrated in Figure 4, the Boost Converter within the Power System is itself made up of interacting components. At the lowest level, each component is associated with a constraint equation. This transfer function is used to provide the quantitative behavior of that component. It is the interaction of these component constraint equations that provides the overall subsystem behavior.
6. Because of space considerations, we will not detail UNICORN's model directed reasoning or classification problem solving techniques at this time. The reader is directed to the references documented at the end of this paper for a description of these technologies.
7. For an excellent general description of basic blackboard technology, references 7 and 9 are highly recommended by the author.
8. Within UNICORN, these events are produced through the use of object demons, and each blackboard panel is an object within an associative network. Panel objects are grouped together to form individual blackboards.

1. The *blackboard handler.*
2. The *agenda scheduler.*

The function of the blackboard handler is to initialize its assigned panel and to keep it clean and free of stale information during normal processing. Within UNICORN, this knowledge source is modeled as an expert system with its own *local context* and its own set of cleanup scripts. These scripts identify significant blackboard events and specify what must be done when they occur. Each blackboard handler is dedicated to a single panel, and so its *input* and *output levels* are identical. The activities of each blackboard handler as well as those of the other knowledge sources are controlled by an agenda scheduler/(blackboard monitor). Within UNICORN, there exists one agenda scheduler per blackboard.[9] The purpose of this knowledge source is to maintain the *focus of attention* of the other knowledge sources assigned to that blackboard. It performs this task by controlling the administration of the agendas within each blackboard panel. When a knowledge source detects an event or sequence of events about which it is interested and is in a position to act upon, it makes a bid to do so by placing a *Knowledge Source Activation Request (KSAR)* on the agenda associated with its assigned panel. This *blackboard object* describes what the bidding knowledge source intends to do if given control and provides an estimate of the affect that its application will have on the portion of the overall problem solution within its limited scope of control. In addition, also contained within the KSAR is an estimate of how much it will cost the system in terms of CPU utilization, solution time, and/or memory space if the associated knowledge source is allowed to execute. The agenda scheduler uses the information contained within the KSAR, along with its own knowledge, posted control directives from higher level monitors, and possible past experiences with the bidding knowledge source, to accept or reject the bid.[10]

If a bid is rejected by an agenda scheduler, the bidding knowledge source simply waits for another chance to bid again at some later time when its chances of success may be better. If, however, a bid is accepted, the knowledge source is given control and performs the action(s) described in the posted KSAR. The result of this action may be one or more blackboard events within that knowledge source's assigned panel or within another panel of its associated blackboard. The events so produced may cause yet other knowledge sources to become active. Two knowledge sources that cooperate in this manner are said to have *overlapping areas of interest.* For cooperative problem solving to work, each knowledge source must produce something that is of interest to another agent either within its own panel or within another panel in its assigned blackboard.

9. As indicated in Figure 5, UNICORN is composed of more than a single blackboard.
10. Each Agenda Scheduler within UNICORN is given a limited resource budget, which it can use to achieve assigned objectives. Any overdrafts must be covered from surpluses available through other agents.

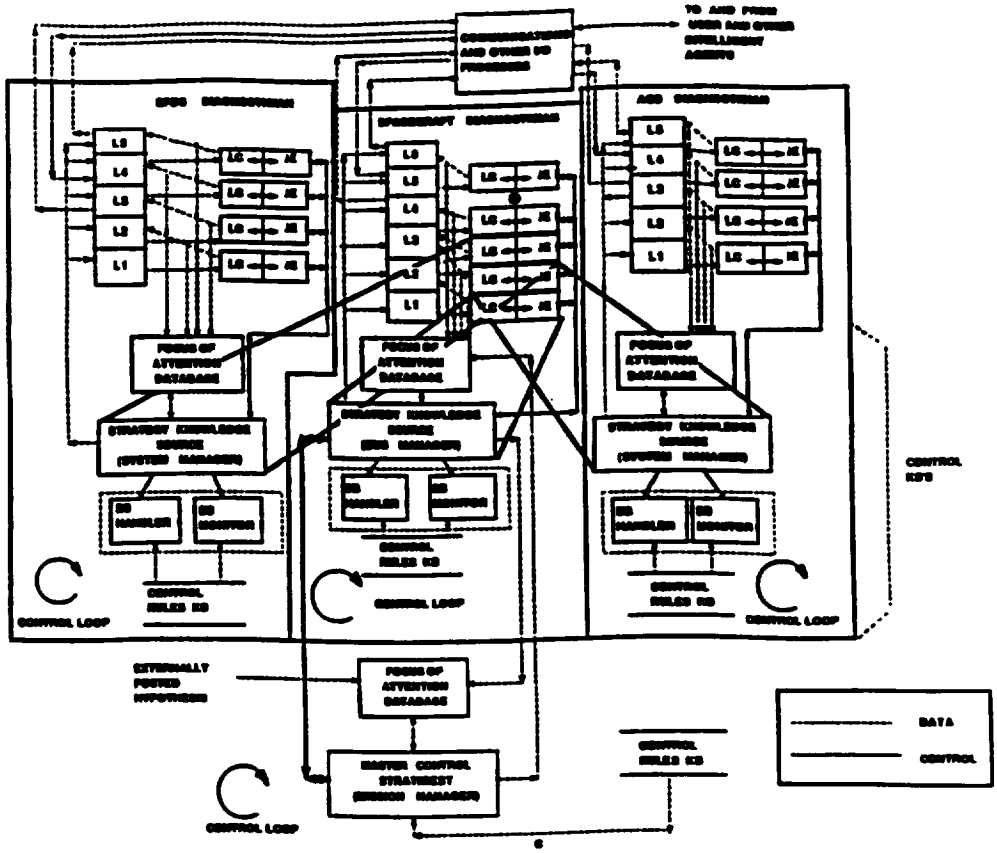


Figure 5. Some UNICORN Blackboards

2.1.3 Thematic Reasoning

Thematic reasoning was first proposed by Rodger Schank as a mechanism for story understanding. In this application domain, themes are utilized to generate expectations about the predicted behavior of an actor (Schank, 1977). Themes can produce these expectations because they can be made to contain background information upon which actor goal predictions can be made. A theme may be programmatically represented as a collection of related goals or even as a generator of these goals. Schank described three types of theme:

1. Role themes
2. Interpersonal themes
3. Life themes

Of these, the first two were extensively utilized within UNICORN. In a role theme, an actor's goals are determined by its role. Once a role is adopted, it sets up expectations about the goals and actions of the role player. Within a blackboard control structure, each knowledge source may be viewed as an actor playing a role. For example, within UNICORN the following actors are resident:[11]

11. These are just the agenda schedulers. There are many other actors, some of which will be introduced in the following sections.

1. ACS Manager
2. EPDS Manager
3. Payload Manager
4. Bus Manager
5. Mission Manager

These actors have specific control related roles, and when a significant event occurs it triggers each to act out that role. For example, the Payload Manager may pass (i.e., MTRANS) a request to the Mission Manager to perform an activity that may in turn result in a resource request of the Bus Manager. Likewise, a fault may be detected within the ACS that could cause the ACS Manager to inform the Bus Manager of a potential problem. The BUS Manager may in turn ask another actor to determine if related symptoms have been detected in the spacecraft subsystem to which it is assigned.

As illustrated in Figure 6, themes are important within UNICORN because they generate the goals that drive the different knowledge sources. These goals are indexed to produce the plans that describe how the identified goal is to be attained. UNICORN utilizes theme related scripts within role objects to determine what actions are required to attain a goal. Any goal may be associated with any number of strategies for its attainment. The strategy selected will depend upon the role being played, the current directive being followed, and the emotional and/or physical context of the actor. An actor may decide to suspend its role in certain circumstances.

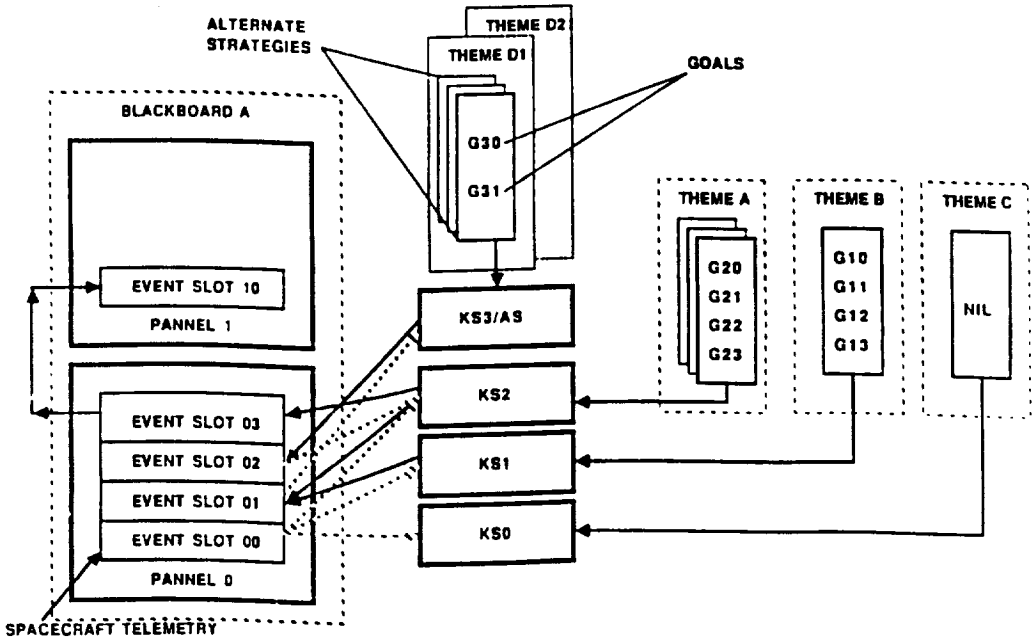


Figure 6. Themes Produce the Goals In Response to Blackboard Events

Role suspension can occur if an interpersonal theme gets invoked. An interpersonal theme consists of a set of test–action pairs where the test is defined as a blackboard event that is indicative of a physical event that threatens the functionality of the spacecraft. The action is the generation of one or more

goals that will in turn produce associated plans that eventually cause a physical change to occur within the spacecraft. For example, an interpersonal theme like *Exhibit–Team–Spirit* can cause an agent like the ACS Manager to give up some or even all of its temporal resources in support of a teammate being threatened with the loss of its own subsystem's functionality. This would be done even though the role of the ACS Manager is to look after only ACS functionality. Thus Thematic Reasoning allows UNICORN to represent control knowledge explicitly, change task priorities, reallocate resources, select KSAR's for execution, and in general react to changes within its dynamic real–time environment.

3.0 SYSTEM FUNCTIONAL DESCRIPTION

The core of the UNICORN system was meant to be placed eventually either on board some future spacecraft or within a ground station wherein it would act as a performance analyst's advisor. Although designed to eventually control all spacecraft functionality, only those object level knowledge sources involved either in diagnosis or related mission management activities have been completed as of this writing. UNICORN's basic external architecture is illustrated in Figure 7. Within this Symbolics–based system, spacecraft and environment behavior is produced through the utilization of quantitative simulators. The environment simulator produces output that is utilized by the spacecraft simulator to produce subsystem behaviors. These are presented to the blackboard process as clock pulses, simulated telemetry packages, and contexts that are unpacked and utilized to produce blackboard events and to drive the diagnostic models. As indicated above, the blackboard paradigm has been utilized to coordinate the activities of a number of knowledge sources and to realize the best problem solutions traded against costs, such as time required to solve a problem and CPU utilization.

In addition, the control structure developed can handle probable cost solution value changes that may result due to temporal aging, physical re–configuration and critical emergency situation changes. To perform these tasks, UNICORN utilized a blackboard structure that was modeled as a conceptual taxonomy in which responsibility is divided along the diagnostic and managerial lines illustrated within Figure 8.

The best way to understand how the blackboard paradigm is utilized within UNICORN is to consider the real–world environment of which it is a model. Within this environment, console operators monitor incoming spacecraft telemetry, scanning certain key observational items for signs of trouble. If these are detected, other normally not actively monitored telemetry points are more closely examined. If a true anomaly is identified, management may request subsystem specialists to further isolate the anomalous behavior to one or more specific subsystems. For critical situations, more than a single expert may be utilized. Each of these individuals may apply a different problem solving technique to arrive at a conclusion. If multiple subsystems are involved, other experts who understand BUS and payload interactions at a much deeper level than do the individual subsystem experts may be consulted. Once a fault is isolated, corrective actions can only be taken after considering

possible inter–system ramifications. Thus while recommendations may be made at lower levels, the final decision as to the corrective action to pursue is made higher in the hierarchy, after review of all options.[12]

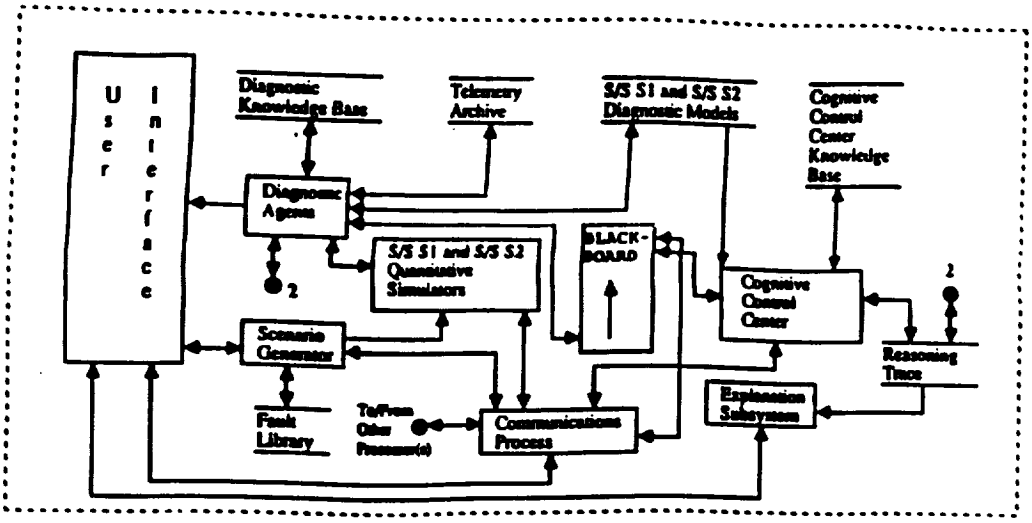


Figure 7. A High Level View Of UNICORN Component Parts

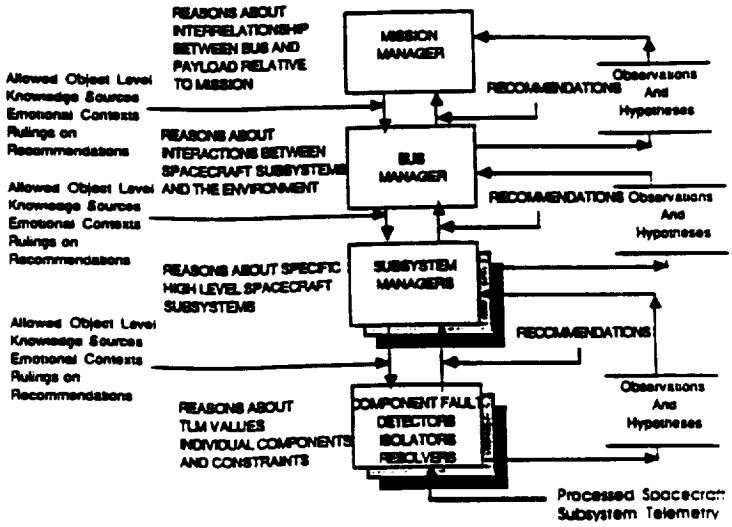


Figure 8. One Way In Which UNICORN's Blackboard Structure May Be Viewed

To support the task flow outlined above, UNICORN's blackboards are arranged hierarchically, as illustrated within Figure 9. At the lowest leaf nodes of this taxonomy are those blackboards associated with the BUS subsystems mentioned in section 2.1.1. Each of these blackboards has been divided into three levels of abstraction corresponding to the different phases of diagnosis (i.e., detec-

12. Actually, the level to which a fault is propagated depends on the estimated impact on mission success. Higher level agents are only made aware of problems if they exceed the scope of control of a lower level agent.

tion, isolation, and resolution/prognosis). Within panel 0 of these blackboards are three symptom detectors. The first is a limit checker, the second is a trend analyzer, and the third uses constraint propagation to identify symptomatic behavior.[13] The first and second utilize fixed context specific limits to screen incoming telemetry. The second requires a number of telemetry frames to detect a symptom while the first needs only one. Both, however, are not usable immediately after a failure, unless constraint relaxation is evoked. The third can be used even after a failure but is CPU intensive and is normally allowed to execute only in situations where the other two can not operate.

Panel 1 of the above–mentioned blackboards has currently assigned to it two fault isolators. The first employs experiential rules to isolate a set of *a priori* defined faults, while the second employs model directed reasoning for fault isolation. The first is not always guaranteed to find the cause of an anomalous behavior, but when it does, it does so relatively quickly. The model directed analyst is more likely to always isolate a fault within its causal model but is CPU intensive and may take a considerable time to isolate certain faults within its domain of application.

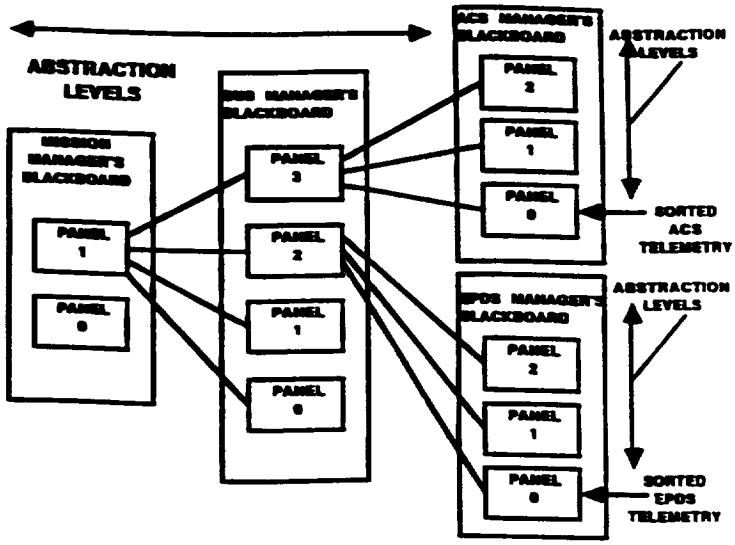


Figure 9. A Taxonomy of UNICORN Blackboards

Panel 2 contains a causal analyst and another agent which performs prognosis. The causal analyst, examines the causal network created within the first two panels and attempts to identify the cause of the fault.[14] All agents within a BUS subsystem associated panel adopt a closed world assumption

13. In addition to the knowledge sources mentioned, each panel is associated with the control level agents described in section 2.1.2.

14. The output of each diagnostic agent is a linked list of hypothesis elements which are arranged within a causal network. This makes the fault hypothesis explicit and readable by the other diagnostic elements. When multiple agents arrive at the same diagnostic conclusion, a numeric confidence level related to that hypothesis is incremented. If it turns out that an agent's hypothesized faults are correct, that agent has its reliability incremented. If it turns out to be wrong, the agent's reliability is decremented.

and assume the fault, if it exists, resides within its own domain of application. The output of this panel consists of one or more recommendations that are passed up to the BUS Manager for review and acceptance or rejection.

Above the BUS subsystem–related blackboards, within UNICORN's blackboard taxonomy, is the BUS Manager's Blackboard. Associated with this blackboard are an number of knowledge sources whose purpose is to reason about faults that seem to cross subsystem boundaries. At this level, the physical environment of the spacecraft is examined as the possible cause of an anomalous behavior. This blackboard is controlled directly by the BUS Manager, whose purpose is to ensure the smooth working of all BUS subsystems. The BUS Manger performs its task by utilizing Thematic Reasoning to determine what goals are necessary and and how they should be pursued. By utilizing a strategy appropriate to the urgency of the situation as seen by itself, its subordinates, and its superior, the BUS Manger can in effect be given an understanding of the importance of its decisions. This is very significant in an environment wherein there may not always be a single answer to a question. Once the BUS Manager, in association with its Health and Maintenance team, has reached a conclusion about what recovery actions are necessary to address a diagnostic situation, it reports them to the Mission Manager.[15]

It is the Mission Manager that is actually given an understanding of mission objectives. This is done by identifying to it which events indicate successful mission accomplishment and which indicate a threat to that objective. The Mission Manager controls the highest level blackboard within the UNICORN blackboard taxonomy. From this vantage point, it receives inputs from, and makes demands, of both the BUS and Payload Managers relative to the performance of their assigned spacecraft systems. Depending upon the nature of the reports received, the Mission Manger may change emotional context and in turn make decisions that result in orders or directives to its subordinates. The Mission Manager may approve or reject a subordinate's recommendations based on its wider understanding of the overall spacecraft operation.

4.0 LESSONS LEARNED

Many lessons have been learned as a result of this effort the following is just a partial list of some of the most interesting ones relative to UNICORN's utilization of the blackboard control paradigm:

1. A blackboard control structure does indeed allow multiple knowledge sources with different problem solving paradigms to work together harmoniously.

15. The BUS Manager is given a time budget by the Mission Manger to arrive at a diagnostic conclusion. It distributes this budget among its subordinate agents. If time appears to be running out, the BUS Manager can halt further investigation and report the current best hypothesis to the Mission Manager. This suggested action will be associated with some degree of confidence. The Mission Manager can than decide to go with the recommendation or else ask for further analysis, if further resources are still available.

2. Local contexts make sense within a blackboard environment, as they provide a knowledge source's own working area for solution derivation.

3. Solutions with higher confidence factors can be produced through cooperative problem solving.

4. A blackboard environment does indeed support distributed knowledge source development.

5. It is vital that the system be hosted on a CPU with sufficient space and horsepower to allow it to operate in a real–time environment.

6. Avoid toy problems, they produce toy solutions.

7. The selection of system goals and task priorities must be allowed to change based on the situation at hand and cannot remain static.

8. Autonomy within a complex system requires the distribution of functionality among multiple agents, each with a world view and scope of control limited to its assigned role.

9. Blackboard construction from scratch is difficult and is no longer recommended.

5.0 FUTURE DIRECTIONS

There are a number of enhancements that would greatly improve the current UNICORN system. The first is to move the system from its current Symbolics environment to a SPARC workstation or equivalent processor. In addition, it is recommended that the KEE–based blackboard architecture be replaced with one based on something like GBB by Blackboard Technology Group, Inc. These modifications should allow the system to run much faster. Second, it is recommended that a case–based reasoner be investigated for addition within the fault isolation panels. These systems could work with the model directed reasoners already in place to acquire new cases automatically. Third, it is recommended that a knowledge source that employs neural network technology be added to each symptom detection panel within UNICORN's blackboards. These knowledge sources could potentially replace the existing limit checking symptom detectors. The possibility of using the current themes contained within UNICORN as a source of diagnostic explanation should be investigated. If themes are good for understanding stories, then it would seem logical that they should also be usable in explaining diagnostic reasoning. Lastly, it might be interesting to go a little further and replace some of the pre–programmed script–based behavior currently employed within UNICORN with dynamically–derived plans produced in an attempt to address an event–triggered objective. Alas, all of this however is for another day.

6.0 CONCLUSIONS

This paper has described one approach to the achievement of system level autonomy. This approach advocates partitive analysis and the distribution of functionality between many intelligent and simi–independent agents, each such agent having a limited world view, but having some understanding of the value of its contributation. In the process of constructing the blackboard control structure, it was discovered that this task was every bit as difficult as constructing the individual diagnostic agents. To shorten the development effort, consideration should be given to the utilization of a commercial blackboard building environment. It was discovered that the concept of Thematic Reasoning allowed for the envisionment of diagnostic agent behavior by making explicit the goals associated with each agent. This explicit representation of functionality helped in the knowledge source construction task.

In the process of constructing this system, many interesting concepts were explored, and much knowledge was gained that will have definite applicability to future efforts in the area of autonomous systems. Much work still remains to be done, however, and many questions as yet remain unanswered.

7.0 ACKNOWLEDGEMENTS

The work documented within this paper was supported with IR&D funding from GE ASTRO Space Division from 1986 through 1990. The author wishes to take this opportunity to thank Robert Schule, Jim White, Chris Sterritt, and Ben Bell for their valuable assistance in helping to bring the future of spacecraft autonomy just a little bit closer to reality.

REFERENCES

1. Abbott K., *Robust Fault Diagnosis Of Physical Systems in Operation*, Phd Dissertation, Rutgers, The State University, 1990.

2. Allen J., *Towards a General Theory of Action and Time*. Artificial Intelligence, 23, 1984.

3. Bell B., Gerner M., and Sterritt C., *DSCS ACS Diagnostic System*, Technical Information Series, No. 87SDS003, GE Astro–Space Division, December 1986.

4. Clancey W., *Classification Problem Solving*. In National Conference on Artificial Intelligence, 1984.

5. Davis R., *Diagnostic Reasoning Based on Structure and Behavior*. In Daniel G. Bobrow, editor, Qualitative Reasoning about Physical Systems, The MIT Press, 1985.

6. de Kleer J. and Williams B., *Diagnosis of Multiple Faults*, Artificial Intelligence, 32, 1987.

7. Engelmore R., Morgan T., *Blackboard Systems*, Addison Wesley Publishers, 1988.

8. Erman L., *An Environment And System For Machine Understanding of Connected Speech*, Phd Dissertation, Stanford University, 1974.

9. Nii H. P., Anton J. J., *Signal–to–Symbol Translation: HASP/SIAP Case Study*, The AI Magazine, Spring 1982.

10. Rossomando P., Sterritt C., Johnson D., *Model Based Diagnosis In An Analog Spacecraft Domain*, Technical Information Series, No. 87SDS043, Astro–Space Division, 05 January 1988.

11. Rossomando P., Bell B., Sterritt C., *UNICORN: Spacecraft Diagnosis in a Distributed Problem Solving Domain*, Technical Information Series, No. 88SDSD002, Astro–Space Division 1989.

12. Schank R., Abelson R., *Scripts, Plans, Goals, and Understanding*, Lawrence Eribaum Associates, Inc. Publishers, 1977.

N92-23365

Intelligent Fault Isolation and Diagnosis for Communication Satellite Systems

Donald P. Tallo, John Durkin
The University of Akron
Akron, Ohio
and
Edward J. Petrik
NASA Lewis Research Center
Cleveland, Ohio

ABSTRACT

NASA-Lewis Research Center recently completed the design of a Ka-band satellite transponder system, as part of the Advanced Communication Technology Satellite (ACTS) System. To enhance the reliability of this satellite, NASA funded The University of Akron to explore the application of an expert system to provide this satellite with autonomous diagnosis capability. The result of this research was the development of a prototype diagnosis expert system, called FIDEX (Fault Isolation and Diagnosis EXpert).

FIDEX is a frame-based system that uses hierarchical structures to represent such items as the satellite's subsystems, components, sensors, and fault states. This overall frame architecture integrates these hierarchical structures into a lattice that provides a flexible representation scheme and facilitates system maintenance. To overcome limitations on the availability of sensor information, FIDEX uses an inexact reasoning technique based on the incrementally acquired evidence approach that was developed by Shortliffe during his MYCIN project. The system is also designed with a primitive learning ability through which it maintains a record of past diagnosis studies. This permits it to search first for those faults that are most likely to occur. And finally, FIDEX can detect abnormalities in the sensors that provide information on the transponder's performance. This ability is used to first rule out simple sensor malfunctions.

The overall design of the FIDEX system, with its generic structures and innovative features, makes it an applicable example for other types of diagnostic systems. This paper discusses these aspects of FIDEX, and illustrates how they can be applied to fault diagnostics in other types of space systems.

Key Words: Expert System, Space Systems, Communication Satellite Systems, FDIR Diagnostics, Frame-Based, Abstract Reasoning, Learning, Sparse Sensors, Sensor Validation

1.0 INTRODUCTION

The satellite network of the United States supports both the commercial and military sectors by providing an effective worldwide communication network. The reliability of this network represents a strategic resource for this country and a critical concern for the National Aeronautics and Space Administration (NASA). Since the mid 1980's, NASA has been investigating the application of expert system technology as a means for improving satellite reliability. The principle motivation for such work has been to develop an intelli-

gent expert system that could be placed onboard a satellite, permitting the satellite to perform autonomous diagnosis. Success in this effort would offer the potential of significantly improving the reliability of satellite communication systems.

In the summer of 1988, NASA-Lewis Research Center funded The University of Akron to study the application of such a diagnosis expert system.

1.1 Overview of Application Area

NASA has recently completed the design of a Ka-band (30/20-GHz) communication satellite transponder. This transponder system is to be integrated within the Advanced Communication Technology Satellite (ACTS) System and deployed early in 1993.

The ACTS transponder is a multiple channel repeater that relays microwave communication signals between highly localized ground terminals; see Figure 1.1. All references to the transponder in this paper are directed towards the components of the communication system that will reside onboard the satellite.

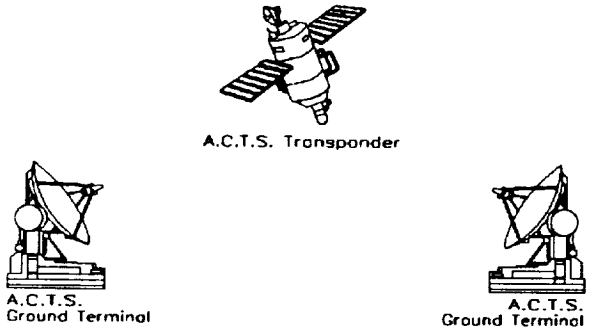


Figure 1.1 ACTS System

Figure 1.2 shows a schematic representation of the ACTS transponder. At present, only two of the multiple channels are implemented in its design. However, this proof of concept design can easily be expanded to incorporate additional links as the system design progresses.

At present, the design of this transponder is being evaluated within the System Integration, Test, and Evaluation (SITE) testbed at NASA-Lewis. The SITE laboratory is used by NASA for validating designs and demonstrating the capabilities of satellite communications systems. This phase of development is valuable to NASA for refining the response of the various systems onboard the transponder. Another important aspect of SITE is the formulation of an understanding of these systems' fault response.

1.2 Project Definition

The goal of this research project was to investigate the possibility of representing the knowledge gained during this SITE phase in a diagnostic expert system. Such a study would then help to lay groundwork for a future system capable of providing the transponder with autonomous diagnosis capability.

The research for this project progressed according to several key developmental phases:

1. *Domain Analysis*: Study the operation of the application system under both normal and abnormal conditions
2. *Knowledge Acquisition*: Study and organize the knowledge used by the domain experts who perform fault diagnostics on application system
3. *Knowledge Representation*: Design a scheme to model the application system and represent the knowledge required to detect, isolate, and diagnose its fault states
4. *Response Strategy Definition*: Establish response strategies and procedures for all fault states

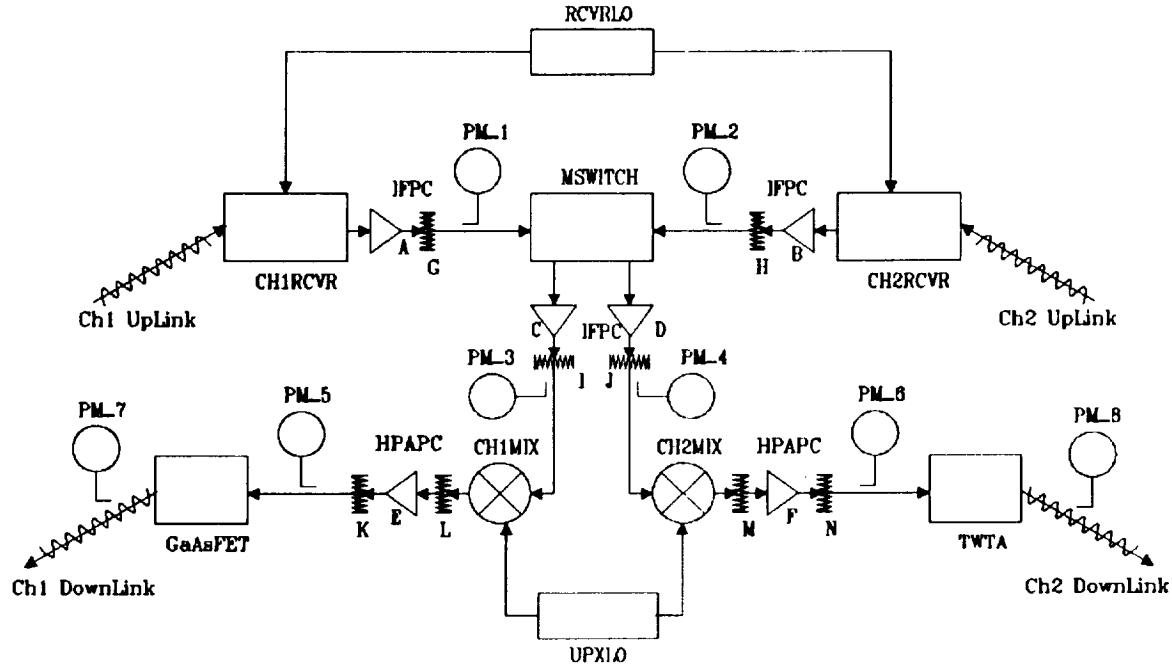


Figure 1.2 SITE Model of the ACTS Transponder System

5. *Prototype Development*: Develop, test, and modify a series of evolutionary diagnostic expert systems
6. *Requirements Definition*: Define the overall specifications for the final diagnostic expert system
7. *Final Development*: Design, encode, integrate, test, and document the deliverable expert system
8. *Life Cycle Analysis*: Define and specify a maintenance schedule for the deliverable diagnostic expert system

During these phases of development, several problems were encountered that reshaped the requirements of the project. Three problems of particular interest resulted from the evolutionary state of the ACTS transponder system. The requirements that these difficulties added to the project, and their solutions, highlight the major strengths of this expert system.

The first of these difficulties became evident during domain analysis. The expert system was constrained to work with limited information on the operational condition of the transponder. Specifically, there were only a few sensors available to provide information on the response of the transponder system. This information was limited to the signal power level sensors, indicated in Figure 1.2 as *PM_1* through *PM_8*, and a few bit error rate (BER) registers. This limited information was not completely adequate for assessing the condition of the transponder. In short, the sensors in the transponder were sparse in number, compared to the other components of the transponder system. Therefore, the isolation of a fault to a specific component based upon sensory information alone was not possible. This limitation was termed the *Sparse Sensor Problem*.

This problem also placed a high premium on the reliability of sensory information. Inconsistent or erroneous readings could render the expert system inoperable. Therefore, a method for resolving conflicts in sensory data was needed.

A second problem was encountered during knowledge acquisition. A prerequisite for the development of any expert system is an extensive understanding of the application area. In a diagnostic application, this requirement dictates that the potential fault states of the system be well known. However, the ACTS transponder was still under evaluation, and a complete understanding of its fault response had yet to be formulated. This fact constrained the investigators to work with limited diagnostic knowledge. Without a clear definition of the transponder's fault response, explicit diagnostic rules were not possible. Therefore, the expert system was prescribed to work with abstract, rather than concrete, diagnostic knowledge.

The final problem was also a result of the evolutionary state of the transponder system. The problem was that changes in the design of the system were always possible. These changes could range from modifications to design specifications, or even the addition of new modules. This situation made it difficult to develop a robust diagnostic agenda.

Faced with these problems, the goal of this project changed more towards a study effort. Emphasis was placed on the development of techniques that would overcome these problems and permit the expert system to reason intelligently with only limited information. The system's knowledge needed to be structured such that any change in the design of the transponder could easily be reflected in the structure of the expert system. All of these requirements placed a premium on the design of knowledge representation techniques and reasoning methods that were general and flexible. The result of this effort was the development of a prototype diagnostic expert system called FIDEX, Fault Isolation and Diagnosis EXpert. This project demonstrated the feasibility of developing an intelligent computer diagnostic system not only for the ACTS transponder, but for space systems in general.

1.3 General Approach to Solution

The general approach taken in the development of this project followed the problem-solving approach used by the ground personnel who perform satellite diagnostics. This strategy was termed the *Modular Approach to Diagnostics*. In general, it follows the four tasks defined below.

1. *Fault Detection*: Monitor the response of the transponder to determine whether it is functioning properly or not
2. *Fault Isolation*: Narrow the range of suspected components to the smallest possible group
3. *Fault Diagnosis*: Investigate the precise nature of the misbehavior and determine the component causing it
4. *Fault Response*: Respond to the diagnosis in a robust and intelligent manner

The purpose of the first task, *Fault Detection*, is to detect any misbehavior in the transponder performance. This task involves the analysis of current sensor information to ascribe qualitative descriptions to each sensor's reading; either *"GOOD"* or *"BAD."* These descriptions are based on whether the data reported by a sensor exceed a tolerance figure centered on its nominal or expected value. Sensor readings that are within tolerance receive a *"GOOD"* description, and those that exceed their tolerance range are labeled as *"BAD."* The detection of a fault is based upon establishing a *"BAD"* reading on any sensor. This indicates that a misbehavior exists in the transponder system and causes the next task to begin.

The second task in this approach is *Fault Isolation*. Its purpose is to isolate the suspected fault to the smallest possible group of components in the transponder. This is accomplished through a principle known as *Error Propagation*. This principle states that the observable symptoms of a misbehavior in a component will propagate through all subsequent sensors in a signal path. The source of such a misbehavior can thus be concluded to lie in that signal path, prior to the detection of the misbehavior, and subsequent to the last sensor indicating a proper signal response.

To implement this, the isolation task considers the qualitative description of all sensor readings as ascribed by the detection phase. It locates a sensor reporting a *"GOOD"* reading that is followed by a *"BAD"* reading. However, because of the sparse sensor limitations, this approach can only isolate the source of the misbehavior to the group of components between these two sensors. For the purposes of this project, these groups of components are termed *SubSystems*, and are defined as the groups of components bounded by signal power level sensors.

The fault isolation task relies heavily upon the integrity of the data reported by the sensors. Should any sensor report erroneous data, this task will fail to reach a valid conclusion. Therefore, a subordinate *Sensor Validation* task was added to this diagnostic phase.

The sub-task of sensor validation is designed to identify the possibility of a faulty sensor. This ability permits the FIDEX system to avoid the search for a non-existent transponder fault. Sensor validation is also based on error propagation; however, in a slightly different fashion. Again, a signal producing a *"BAD"* sensor reading at one point in the transponder should result in a *"BAD"* reading on all subsequent sensors in that signal path. This task identifies the possibility of a faulted sensor if a *"GOOD"* reading instead is found.

In either case, the purpose of isolation is to identify the subsystem containing the component causing the misbehavior. If this misbehavior is the result of a component failure, the subsystem identified by its input and output sensor readings is flagged as isolated. However, if the detected *"BAD"* sensor reading is the result of a faulty sensor, isolation flags the sensory components as the isolated subsystem. Once the source of the fault is isolated, the next task is initiated.

The third task, *Fault Diagnosis*, involves consulting a community of diagnostic expert systems. Each system is designed to address the problems of a specific subsystem within the transponder. Determining the appropriate diagnostic expert to be consulted is the final task of the isolation phase.

These specialized diagnostic systems use knowledge that is rule-based and backward chaining in nature. The hypotheses for these rules represent the potential faults in the isolated subsystem. The order in which they are placed on the agenda is based on the history of the fault states. Maintaining this history permits FIDEX to pursue the most likely problems first.

Each diagnostic system was also designed with an ability to perform inexact reasoning. This was done to overcome problems that resulted from limited

information about the transponder's performance. Such an ability was important in that the FIDEX system would often need to make a "guess" at the most likely fault state.

The inexact reasoning technique chosen for this project was based on the certainty theory given by Shortliffe (1975), with some modification by Durkin (1991). It relies upon establishing incremental measures of belief or disbelief in rule conclusions. These two factors are then used to establish an overall confidence when a conclusion is supported by multiple rules.

The final task is *Fault Response*. The present strategy for fault response is to provide recommendations for reconfiguring the components or sensors. Plans are to include the capability to reconsider fault diagnosis if the recommended action was ineffective. FIDEX would retain its past diagnosis, including recommendations, and reconsider the problem with information made available following the corrections to the transponder.

The remainder of this paper discusses the workings of the FIDEX system. It will demonstrate the techniques discussed above and, by example, show their application to other types of diagnostic systems.

KNOWLEDGE REPRESENTATION

The diagnostic knowledge of FIDEX is represented using both frame-based and rule-based techniques. This section discusses the structure of that hybrid framework. It also provides sample code segments describing the actual implementation in the syntax of NEXPERT Object, the software development tool used in the project.

2.0 FRAME NETWORK STRUCTURE

The expert system needed to be designed such that it would easily allow the incorporation of changes to the transponder. Therefore, it was decided that a frame-based approach for knowledge representation would be appropriate.

Frame hierarchies were developed to represent the transponder's components, subsystems, sensors, and fault states. These hierarchies were interconnected into a network to enrich the overall knowledge representation structure.

2.1 Structure of Components Class

A frame hierarchy was created to provide a clear and efficient representation of all components in the transponder. Figure 2.1 shows this structure called the *Components Class*. This figure illustrates a convention that will be maintained throughout in this paper. Circles represent class frames and triangles represent object frames. Lines indicate links between frames, with the arrows indicating the direction of inheritance.

The root node in Figure 2.1 is a circle indicating a class frame called *Components*. This class was created to represent the commonality between all components in the transponder. It is divided into several subclasses represented by the second level of class frames. Each of these subclasses describes the function of components in the transponder: amplifiers, attenuators, etc. The components are represented by object frames attached to these subclasses.

The code segment describes this structure. The first series of declarations defines the properties that are to be used. This is not

a complete listing. Only the properties of interest in this discussion are shown.

Properties were defined to describe physical characteristics about a component: its name, input/output components, etc. These properties are used by FIDEX to give a component a self awareness. Other properties provide functional information about the components: its input and output signal power levels, gain, nominal gain, etc.

The next definition creates a class frame called *COMPONENTS* in the object space of the expert system. It establishes links to several subclasses and defines which properties will be associated with this class. Each subclass inherits all properties associated with the *COMPONENTS* class. Any properties specific to a type of component can be defined at the subclass level. The definition for the *ATTENUATORS* class is included as an example of this. The *SETTING* property is used to describe the variable attenuation setting of the attenuators in the transponder system.

```
(@PROPERTY = COMPONENTINN     @TYPE = String;)
(@PROPERTY = COMPONENTOUT     @TYPE = String;)
(@PROPERTY = FAILED           @TYPE = Boolean;)
(@PROPERTY = GAIN             @TYPE = Float;)
(@PROPERTY = GAINNOMINAL      @TYPE = Float;)
(@PROPERTY = NAME             @TYPE = String;)
(@PROPERTY = SETTING          @TYPE = Integer;)
(@PROPERTY = POWERINN         @TYPE = Float;)
(@PROPERTY = POWEROUT         @TYPE = Float;)

(@CLASS = COMPONENTS
          (@SUBCLASSES = AMPLIFIERS
                         ATTENUATORS
                         BERREGISTERS
                         GaAsFETS
                         LOCALOSCILLATORS
                         MIXERS
                         POWERMETERS
                         RECEIVERS
                         SWITCHES
                         TWTAS )
          (@PROPERTIES = COMPONENTINN
                         COMPONENTOUT
                         FAILED
                         GAIN
                         GAINNOMINAL
                         NAME
                         POWERINN
                         POWEROUT ) )

(@CLASS = ATTENUATORS
          (@PROPERTIES = SETTING ) )

(@OBJECT = IFPCATTEN1
           (@CLASSES = ATTENUATORS ) )
```

Finally, the last definition in the code segment shows the attachment of an object frame to this structure. An object frame called *IFPCATTEN1* is created in the object space to represent one of several *IF* signal *P*ower level *C*ontrol *ATTEN*uators in the transponder. This attenuator object is assigned to the *ATTENUATORS* class. Therefore, it inherits all properties assigned to both this class and the *COMPONENTS* class. Each component of the transponder is represented by an object frame in this manner.

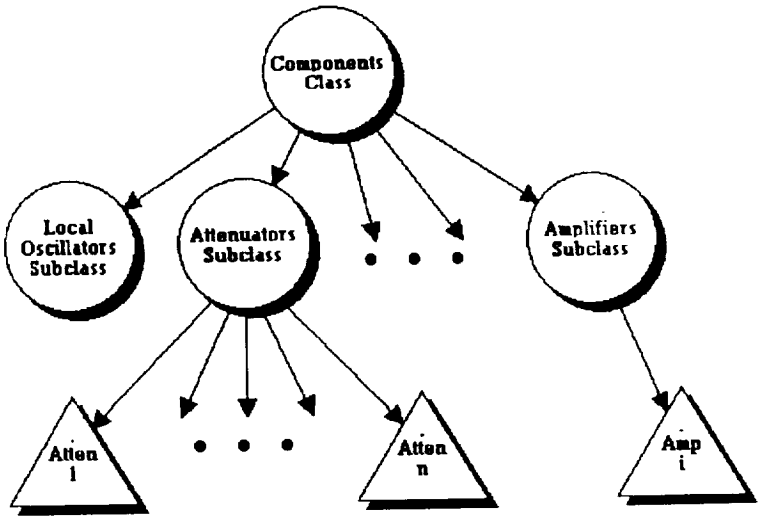


Figure 2.1 Components Class

2.2 Structure of Subsystems Class

Each component is also associated with a subsystem of the transponder (see Figure 2.2). Several object frames are used to represent the collections of components called subsystems. These frames are then organized by attaching them to a class frame for all subsystems in the transponder. Finally, the membership of a component in a particular subsystem is represented by attaching its object frame as a subobject of the appropriate subsystem object frame.

Again, a code segment is provided to describe this structure. Similar to the components definition, several properties are defined to represent both structural and functional information about the subsystems of the transponder.

A class frame called *SUBSYSTEMS* is created in the object space of the expert system. Properties assigned to this class are inherited by all attached object frames. Finally, the last definition in the code segment shows the assignment of an object frame to this structure.

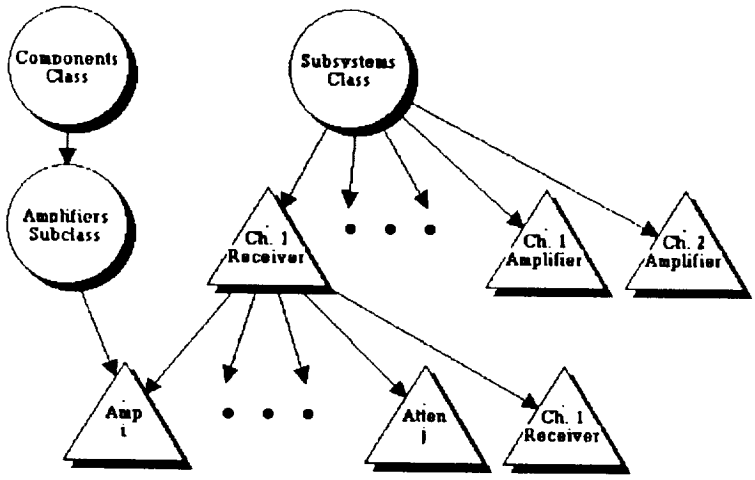


Figure 2.2 Subsystems Class

An object frame called *CH1RECEIVER-SYSTEM* is created in the object space to represent the group of components associated with the Channel 1 Receiver Subsystem. Object frames to represent the Channel 1 Receiver unit, an IF Signal Power Control Amplifier and Attenuator, and the Receiver Local Oscillator are attached as subobjects of this subsystem.

```
(@PROPERTY = ISOLATED          @TYPE = Boolean;)
(@PROPERTY = READINGINN        @TYPE = String;)
(@PROPERTY = READINGOUT        @TYPE = String;)
(@PROPERTY = SYBSYSTEMINN      @TYPE = String;)
(@PROPERTY = SUBSYSTEMOUT      @TYPE = String;)

(@CLASS = SUBSYSTEMS
          (@PROPERTIES = GAIN
                         GAINNOMINAL
                         ISOLATED
                         NAME
                         POWERINN
                         POWEROUT
                         READINGINN
                         READINGOUT
                         SUBSYSTEMINN
                         SUBSYSTEMOUT ) )

(@OBJECT = CH1RECEIVERSYSTEM
           (@CLASSES = SUBSYSTEMS )
           (@SUBOBJECTS = CH1RCVR
                          IFPCAMP1
                          IFPCATTEN1
                          RCVRLO ) )
```

As these frames represent components of the transponder, they are attached to the *COMPONENTS* class structure as well. This linking of component object frames to the components world can be interpreted as an *Is-A Link*. Links to the subsystems world represent *Part-Of Links*. That is, the IFPC Amplifier *Is A* n amplifier and is *Part Of* the Channel 1 Receiver system.

This approach not only aids the diagnostic tasks, but also provides an efficient coding approach. Through multiple inheritance, each subsystem component acquires information from two parents. One

provides information on performance while the other provides information on structure.

2.3 Structure of Sensors Class

Two types of sensory element monitor both the response of the transponder and the relayed signal. The first type is signal power level sensor. The other type represents the data stream bit error rate (BER) registers located within the ground terminal systems. The information used for diagnosis is provided by these sensors. These sensors were represented by creating the class structure for all sensory components shown in Figure 2.3.

This structure is divided into subclasses according to the two types of sensor. Each sensor is then represented by an object attached to the appropriate type subclass. The code segment creates this structure in the object space of the expert system.

Properties are defined to describe the *DATA* reported by a sensor, its *NOMINAL* value, the corresponding *ERROR*, and the *TOLERANCE* band of acceptable error magnitudes. A string property called *READING* is used for the qualitative descriptions that were introduced in section 1.3.

```
(@PROPERTY = DATA        @TYPE = Float;)
(@PROPERTY = ERRORR      @TYPE = Float;)
(@PROPERTY = NOMINAL     @TYPE = Float;)
(@PROPERTY = READING     @TYPE = String;)
(@PROPERTY = TOLERANCE   @TYPE = Float;)

(@CLASS = SENSORS
          (@SUBCLASSES = BERSENSORS
                         POWERSENSORS )
          (@PROPERTIES = DATA
                         ERRORR
                         NAME
                         NOMINAL
                         READING
                         TOLERANCE ) )
 (@CLASS = BERSENSORS
          (@SUBCLASSES = CH1BERSENSORS
                         CH2BERSENSORS ) )
 (@CLASS = COMPONENTS
          (@SUBCLASSES = BERREGISTERS
                         POWERMETERS ) )
(@OBJECT = BER1
          (@CLASSES = BERREGISTERS
                      CH1BERSENSORS ) )
(@OBJECT = PM1
          (@CLASSES = POWERMETERS
                      POWERSENSORS ) )
```

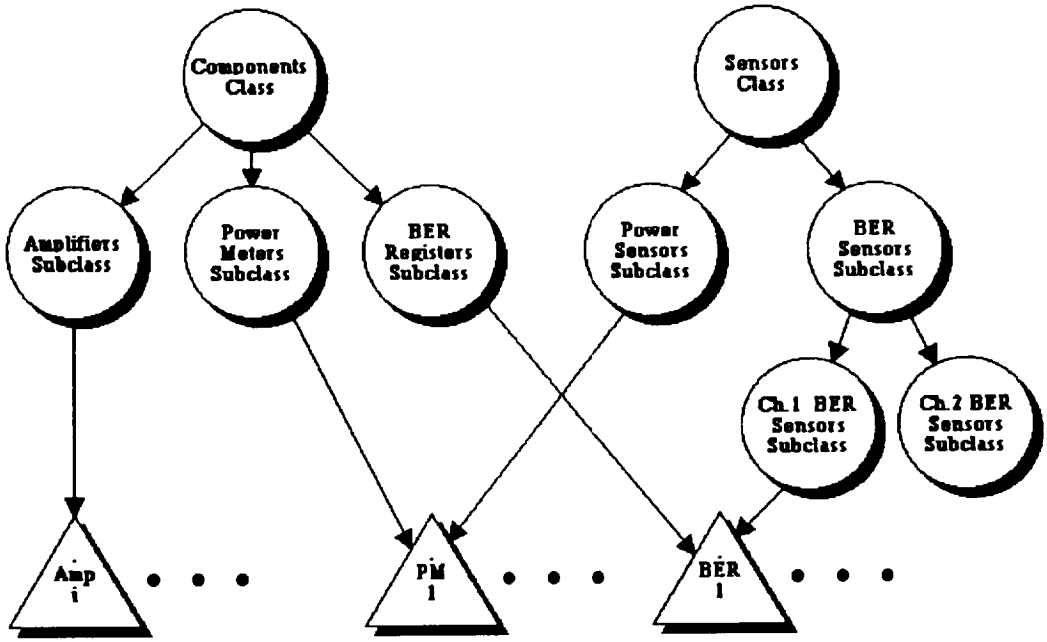


Figure 2.3 Sensors Class

The *BER SENSORS* class is also divided into two subclasses according to their channel. This was done to simplify the analysis of frequency-dependent fault states. It also demonstrates how class structures can be cascaded to further describe component function and organization.

Like all other transponder components, sensory elements could potentially fail. Therefore, each sensor is also represented in FIDEX as a member of the component world. The code segment shows the definition of two sensor type subclasses in the *COMPONENTS* world.

Each sensory component is represented by an object frame. The example shows the definition of one BER sensor, *BER1*, and one signal power level sensor, *PM1*. These frames are linked to their appropriate type subclass in both the components world and the sensors world.

2.4 Structure of Fault States Class

The transponder fault states are represented as objects in a class structure called Fault States. This class is also divided into several subclasses. Each subclass frame represents the association of fault states to component types: amplifier faults, attenuator faults, etc. Object frames representing the specific failure modes of the transponder are then attached to the appropriate subclasses. This structure, shown in Figure 2.4, enables FIDEX to reason about both known and abstract faults.

The code segment that defines this structure is nearly identical to that of the *COMPONENTS* class. This is because the type of fault state is associated with the type of component.

The primary properties associated with the *FAULT STATES* class are listed first. These describe which *COMPONENT* the fault is associated with, its *INFeRence CATEGORY* or priority, and the *POWER SYMPTOM GROUP* with which it is associated. A boolean property, *VERIFIED*, is used to flag fault states that have been verified by the diagnostic process. The final property listed is *VALUE*. This property is reserved by NEXPERT. The fault states represent the hypotheses of rules used during diagnosis. This property is assigned the results of rule evaluations.

The diagnostic process reasons with the fault state hypotheses using two distinct techniques. The next section discusses these, and provides structural information on their implementation.

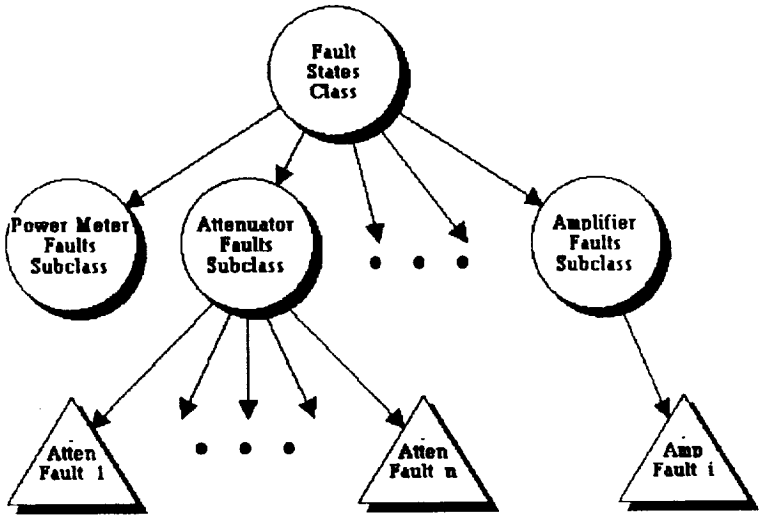


Figure 2.4 Fault States Class

```
(●PROPERTY = COMPONENT                ●TYPE = String;)
(●PROPERTY = INFRCATEGORY             ●TYPE = Integer;)
(●PROPERTY = POWERSYMPTOMGROUP        ●TYPE = String;)
(●PROPERTY = VERIFIED                 ●TYPE = Boolean;)
(●PROPERTY = Value                    ●TYPE = Boolean;)

(●CLASS = FAULTSTATES
          (●SUBCLASSES = AMPLIFIERFAULTS
                         ATTENUATORFAULTS
                         BERREGISTERFAULTS
                         GaAsFETFAULTS
                         LOCALOSCILLATORFAULTS
                         MIXERFAULTS
                         POWERMETERFAULTS
                         RECEIVERFAULTS
                         SWITCHEFAULTS
                         TWTAFAULTS )
          (●PROPERTIES = COMPONENT
                         INFRCATEGORY
                         NAME
                         POWERSYMPROMGROUP
                         VERIFIED
                         Value ) )

(●OBJECT = IFPCAMPSUPPLYFAILURE
           (●CLASSES = AMPLIFIERFAULTS ) )
```

3.0 REASONING TECHNIQUES

FIDEX reasons with two distinctly different techniques. The first technique, *Absolute Reasoning*, is used to establish or reject the existence of concrete, pre-defined fault states. The second technique, *Abstract Reasoning*, is used to recover when the diagnostic task cannot reason effectively using the first technique. Under such conditions, the second technique is used to establish evidence in conceptual fault states.

3.1 Absolute Reasoning

In general, knowledge that supports rules in absolute terms is *Associative Knowledge*. This type of knowledge associates conditions with the establishment or rejection of a conclusion. FIDEX uses two types of associative knowledge.

The first type is *Directly Associative*. This knowledge directly associates conditions with conclusions. An example of this type of knowledge might be: *If the data reported by a sensor reading exceeds its tolerance band, then the sensor's reading is "BAD."*

The condition of sensor data exceeding its acceptable range is directly associated with establishing a *"BAD"* qualitative description for that reading. Rules that represent this type of knowledge are used to structure the strategies of the diagnostic tasks.

However, the majority of the knowledge used in the task of fault diagnosis is supported by an accumulation of evidence. This type of knowledge is *Cumulatively Associative*. That is, the accumulation of several conditions is associated with the establishment or rejection of a conclusion. Moreover, each condition may contribute differently to that conclusion. An example of such knowledge might be: *A LOW signal power level might indicate internal phase lock failure in a local oscillator,* and *A HIGH bit error rate might indicate that the local oscillator is out of phase lock.*

Neither condition can be directly associated to establish or reject the conclusion of an internal phase lock failure. However, each contributes evidence to that conclusion. When multiple rules contribute evidence toward a conclusion, the system must be able to accumulate this evidence. The FIDEX system has such an ability.

3.2 Incremental Accumulation of Evidence

FIDEX uses the *Incremental Accumulation of Evidence* to establish or reject hypotheses that are supported by multiple rules. The technique used by FIDEX follows the work done by Shortliffe (1975) in his MYCIN project, with some modifications by Durkin (1991).

The first two equations given below accumulate a measure of belief, AB, and disbelief, AD, in a hypothesis, H. These

two measures are then used by the third equation to establish an overall confidence, *CF*, in that hypothesis. These equations work as follows.

$$AB(H)_k = AB(H)_{k-1} + MB(H)_k \cdot [\ 1 - AB(H)_{k-1}\ ]$$

$$AD(H)_k = AD(H)_{k-1} + MD(H)_k \cdot [\ 1 - AD(H)_{k-1}\ ]$$

$$CF(H)_k = [\ \frac{AB(H)_k - AD(H)_k}{1 - \min(AB(H)_k, AD(H)_k)}\ ]$$

Rules that accumulate knowledge do not assign boolean values to their associated conclusions. Instead, they determine a measure of belief, *MB*, or measure of disbelief, *MD*, in that conclusion. These measures represent the degree to which the conclusion of that rule has contributed to the establishment or rejection of its hypothesis. The values that are assigned to these measures range between 0 and 1. Values close to 1 represent strong measures while values close to 0 represent weak measures. A value of 1 is generally not assigned, as it results in a boolean value for *AB* or *AD*.

Consider an arbitrary hypothesis, *H*, and assume that no evidence has been established toward belief in that conclusion, $K = 0$ and $AB(H)_0 = 0$. Establishing a fact in support of this conclusion might assign a measure of 0.2 to the belief in *H*, $MB(H)_1 = 0.2$. The accumulated belief in the hypothesis would then be: $AB(H)_1 = 0.2$. The establishment of another piece of evidence in support of *H* might assign a measure of 0.5 to the belief in *H*, $MB(H)_2 = 0.5$. The accumulated belief in the hypothesis would then be incremented: $AB(H)_2 = 0.6$.

The accumulated measure of disbelief, *AD(H)*, is incremented similarly. However, this accumulation would be founded on rules that establish measures of disbelief in a hypothesis, $MD(H)_k$. This measure indicates evidence in opposition of the hypothesis.

As rules ascribe $MB(H)_k$'s and $MD(H)_k$'s, and accumulated values are calculated, the overall confidence in a conclusion, $CF(H)_k$, is calculated. Confidence factors range in value from -1 to 1. A value near -1 signifies little confidence in the conclusion, or the rejection of the hypothesis. A value near 1 denotes a high level of confidence, or the establishment of the hypothesis. Values in between represent various degrees of confidence, with 0 meaning unknown.

The following segments of code implement this technique. First, properties are defined to represent the *MB*, *MD*, *AB*, *AD*, and *CF* values required by the method. A string for a qualitative description of *CONFIDENCE* in a hypothesis is also defined.

```
(@PROPERTY = AB          @TYPE = Float;)
(@PROPERTY = AD          @TYPE = Float;)
(@PROPERTY = CF          @TYPE = Float;)
(@PROPERTY = CONFIDENCE  @TYPE = String;)
(@PROPERTY = MB          @TYPE = Float;)
(@PROPERTY = MD          @TYPE = Float;)

(@CLASS = CERTAINTYANALYSIS
          (@PROPERTIES = AB
                         AD
                         CF
                         CONFIDENCE
                         MB
                         MD ) )

(@OBJECT = IFPCAMPSUPPLYFAILURE
           (@CLASSES = AMPLIFIERFAULTS
                       CERTAINTYANALYSIS ) )
```

Next, a class for *CERTAINTY ANALYSIS* is defined and assigned these properties. All frame objects that require certainty analysis can then be attached to this class frame per the *IFPCAMP SUPPLY FAILURE* fault state shown in the example.

The primary purpose of this class structure is to provide the overhead required to

ascribe qualitative descriptions for *CONFIDENCE* in a hypothesis. The inference process for this assignment is triggered by active facets associated with these properties.

Methods are assigned to the *MB*, *MD*, *AB*, *AD*, and *CF* properties, and inherited by all object frames attached to this class. Three types of method are used. The first type, *Initial Value*, defines parameters to be used to initialize property values on reset or initialization of the inference process. The second type, *Order of Sources*, defines procedures to be taken to establish property values during the inference process. The final type, *Change Actions*, provides procedures to be followed when a property value changes.

In the syntax of NEXPERT, such methods associated with properties are called "meta-slots." The following code segment defines the meta-slots assigned to *CERTAINTY ANALYSIS* properties. They are inherited by all object frames that are attached to this class.

```
(●SLOT = CERTAINTYANALYSIS.AB
       (●INITVAL = 0.0)
       (●SOURCES = (RunTimeValue (0.0)))
       (●CACTIONS = (Do ((SELF.AB - SELF.AD) /
                         (1 - min(SELF.AB, SELF.AD))) (SELF.CF)))

(●SLOT = CERTAINTYANALYSIS.AD
       (●INITVAL = 0.0)
       (●SOURCES = (RunTimeValue (0.0)))
       (●CACTIONS = (Do ((SELF.AB - SELF.AD) /
                         (1 - min(SELF.AB, SELF.AD))) (SELF.CF)))

(●SLOT = CERTAINTYANALYSIS.CF
       (●INITVAL = 0.0)
       (●SOURCES = (RunTimeValue (0.0)))
       (●CACTIONS = (Reset (Evaluate Certainty Factors))
                    (Do (Evaluate Certainty Factors)
                    (Evaluate Certainty Factors))))

(●SLOT = CERTAINTYANALYSIS.MB
       (●INITVAL = 0.0)
       (●SOURCES = (RunTimeValue (0.0)))
       (●CACTIONS = (Do (SELF.AB + SELF.MB * (1 - SELF.AB))
                    (SELF.AB))
                    (Reset (SELF.MB))))

(●SLOT = CERTAINTYANALYSIS.MD
       (●INITVAL = 0.0))
       (●SOURCES = (RunTimeValue (0.0)))
       (●CACTIONS = (Do (SELF.AD + SELF.MD * (1 - SELF.AD))
                    (SELF.AD))
                    (Reset (SELF.MD))))
```

The initial conditions of the attached objects are assured by first setting these property values to 0. In that these objects represent hypotheses, this establishes the state of the system to $K = 0$ at both initialization and run-time.

When a value is assigned to the *MB* or *MD* property of an object in this class, the change actions inherited from these slots will fire. The equations for the accumulation of belief/disbelief are evaluated, and the result assigned to the current object's *AB* or *AD* property. *SELF* denotes the current object in class level definitions. The value of that property is then reset to its run-time value, 0.

With the assignment of a new value to either the *AB* or *AD* property, inherited change actions will fire. These actions evaluate the equation for the confidence factor and assign that value to the objects *CF* property. And, in turn, change actions associated with that slot fire. However, the actions taken by that slot are more complicated because it manipulates the agenda.

The NEXPERT agenda is a prioritized list of hypotheses to be pursued. When hypotheses are placed on the agenda by conventional means, they are pursued in an order defined by their priority, or inference category. However, this protocol can be overridden through the direct assignment of hypothesis to another value.

The purpose of the change actions associated with the *CF* slot is to affect the ascription of a qualitative description to the current object's *CONFIDENCE* property. This requires the evaluation of nine rules corresponding to as many delineations. Each rule has the same hypothesis, *Evaluate Certainty Factors*.

First, the change actions reset the value of this hypothesis to unknown. Then the value of the hypothesis is assigned to itself. Since the value of the hypothesis in unknown, this assignment of the hypothesis to itself forces the evaluation of its supporting rules.

Three of the nine rules that support this hypothesis are given in the following sample code segment. The example demonstrates the ascription of qualitative descriptions for *CF* values near 1 and -1, and an arbitrary range in between.

```
(@RULE = RULE 23 EVALUATION OF CERTAINTY FACTORS
        (@LHS = ( ≥ ( \CURRENT FAULT.NAME\.CF ) ( 0.9 ) ) )
        (@HYPO = Evaluate Certainty Factors )
        (@RHS = ( Let ( \CURRENT FAULT.NAME\.CONFIDENCE )
                      ( "ESTABLISHED" ) )
                ( Let ( \CURRENT FAULT.NAME\.VERIFIED )
                      ( TRUE ) ) ) )

(@RULE = RULE 24 EVALUATION OF CERTAINTY FACTORS
        (@LHS = ( ≤ ( \CURRENT FAULT.NAME\.CF ) ( -0.9 ) ) )
        (@HYPO = Evaluate Certainty Factors )
        (@RHS = ( Let ( \CURRENT FAULT.NAME\.CONFIDENCE )
                      ( "REJECTED" ) )
                ( Let ( \CURRENT FAULT.NAME\.VERIFIED )
                      ( FALSE ) ) ) )

(@RULE = RULE 27 EVALUATION OF CERTAINTY FACTORS
        (@LHS = ( ≥ ( \CURRENT FAULT.NAME\.CF ) ( 0.5 ) )
                ( < ( \CURRENT FAULT.NAME\.CF ) ( 0.75 ) ) )
        (@HYPO = Evaluate Certainty Factors )
        (@RHS = ( Let ( \CURRENT FAULT.NAME\.CONFIDENCE )
                      ( "POSSIBLE" ) ) ) )
```

The conditions, *@LHS*, of *RULE 23* will be true if the value of the *\CURRENT FAULT.NAME*'s *CF* is greater than or equal to 0.9. *CURRENT FAULT* is a blackboard object in FIDEX. Prior to the assignment of any certainty analysis measures, the name of the current fault state is posted to the *NAME* property of this object. This generic rule looks to the blackboard to determine the name of the current fault. Its *CF* is tested and the hypothesis accordingly established or rejected.

If the hypothesis is established, several actions are taken, *@RHS*. These actions first assign a qualitative description of *"ESTABLISHED"* to the *CONFIDENCE* property of the current fault. Then its *VERIFIED* property is set to *TRUE*. These rules are non-exhaustive. Therefore, the firing of one rule will terminate the evaluation of certainty factors.

RULE 24 evaluates a *CF* value at the opposite end of the scale. If this value is less than or equal to -0.9, the confidence in the hypothesis is described as *"REJECTED"* and its verification property is set to *FALSE*.

The final rule, *RULE 27* given as an example, shows the evaluation of a certainty factor in a range between those bounds. This rule states that if the *CF* of the current object is less than or equal 0.75 and greater than 0.5, the confidence in the current fault state is *"POSSIBLE."* This level of confidence is not sufficient to establish or reject the fault. Therefore, no assignment is made to its verification property.

Discussion to this point has been on the incremental accumulation of evidence toward concrete fault states. The next topic will discuss the application of these techniques for abstract reasoning.

3.3 Abstract Reasoning

In general, knowledge that supports rules in abstract terms is *Conceptual Knowledge*. This type of knowledge is *Indirectly Associative Knowledge*. It associates condition to abstract ideas that are indirectly related to the rule being pursued. An example of this type of knowledge might be: *A HIGH bit error rate is typical of a misbehavior in one of the frequency conversion components.*

FIDEX uses this type of reasoning to establish levels of confidence in class level

fault categories. That is, it might reach a conclusion of the form: *The observed symptoms are typical of those associated with a failure of the local oscillator.*

During the diagnostic task, FIDEX exhausts its knowledge about the fault states of the system. It is entirely possible that a failure mode might occur for which FIDEX has no knowledge. In that case, it would resort to confidence accumulated in class level fault states as its diagnostic conclusion.

This abstract reasoning ability of FIDEX is implemented as follows. First, all of the fault state type subclasses defined in section 2.4 are attached as subclasses of the class *CERTAINTY ANALYSIS*. Therefore, they inherit this class overhead.

By doing this, measures of belief and disbelief can also be assigned to the class properties. Levels of confidence can then be accumulated at this class, or conceptual, level. An example of such an assignment is given in the following rule.

This is a directly associative rule that establishes a qualitative description for the bit error rates during diagnostics. One of several hypotheses that are indirectly associated with concluding a *"HIGH"* bit error rate is that this symptom is associated with the class of *LOCAL OSCILLATOR FAULTS*. Therefore, the last two actions of this rule assign a measure of 0.3 to the belief that the fault is associated with a local oscillator.

```
(@RULE = RULE47 HIGH BIT ERRORR RATE
       (@LHS = ( ≤ ( <|CH1 BERSENSORS|>.RATE ) ( 0.01 ) ) )
       (@HYPO = Bit Errorr Rates Are HIGH )
       (@RHS = ( Let ( <|CH1 BERSENSORS|>.RATE ) (/HIGH" ) )
          (Let (CURRENTFAULT.NAME) (// |LOCAL OSCILLATOR FAULTS|"))
          ( Do ( 0.3 ) ( \CURRENTFAULT.NAME\.MB ) ) ) )

  (@SLOT = IFPCAMP SUPPLY FAILURE
           @INFATOM = IFPCAMP SUPPLY FAILURE.INFR CATEGORY; )
```

Using this technique, FIDEX can piece together information and reach conceptual conclusions such as the one given above. The final topic in this section is the representation of FIDEX's learning capacity.

4.0 LEARNING & SEARCH STRATEGY

There are two databases used by FIDEX. One contains information required to initialize parametric values of the system. Each record contains information on nominal readings, error tolerances, and other initial parameters. These values are loaded and stored in the appropriate slots of objects at runtime or when FIDEX is initialized. This method of initialization was chosen to facilitate the maintenance of the system.

The second database is used to provide FIDEX a limited learning capability. FIDEX stores the failure history of the transponder system in this database. Each known fault state is represented by a record that contains fields that represent the failure history of that fault state. Following diagnostics, FIDEX increments the history of the identified fault. This record keeping is used to direct the search strategy of future sessions toward the most likely faults.

The search strategy is adaptive in that the priorities by which known fault states are placed on the agenda is based upon the values maintained in the history database. A class level property of all fault states is the integer *INFR CATEGORY*. The value of this property is retrieved from the database when the diagnostic task is initialized. This property is then assigned to the inference priority of the fault state hypothesis by slot actions. The previous example shows

such a slot for one fault state. All fault state inference atoms are similarly initialized.

When the diagnostic task establishes a known fault state, the value of its inference category is incremented accordingly. The updated value is then stored in the learning database.

5.0 SUMMARY

The prototype FIDEX system is the result of a study effort by The University of Akron, funded by NASA-Lewis Research Center. Its purpose was to demonstrate that expert system technology can be applied to enhance the reliability of satellite communication systems, in particular, the Ka-band Advanced Communication Technology Satellite Transponder.

The initial goal of this research was to develop and expert system to provide this satellite with autonomous diagnosis capability. As limitations prevented the autonomy of FIDEX, the project became more of a study effort. Its goal changed towards the development of techniques to overcome several limiting problems.

The resulting system used hierarchical frame-based structures to represent the structure and operation of the satellite. Other strengths of FIDEX included its use of inexact reasoning techniques, its primitive learning ability, and its capacity for detecting abnormalities in sensors.

The overall design of the FIDEX system made it an applicable example for other types of diagnostic system. This paper discussed these aspects of FIDEX, and illustrated how they could be applied to fault diagnostics in other types of space systems.

REFERENCES

Durkin, J, Tallo, D.P., Petrik, E.J., *"FIDEX: An Expert System for Satellite Diagnostics,"* Space Communications Technology Conference Onboard Processing and Switching, NASA Conference Publication 3132, Cleveland, Ohio, November 12-14,(1991)

Harmon, P., Maus, R., Morrissey, W., Expert Systems Tools & Applications , John Wiley & Sons, New York, New York (1988)

Kerczewski, R.J., Fujikawa, G., *"Performance Measurements for a Laboratory-Simulated 30/20 GHz Communication Satellite Transponder,"* 13th AIAA International Communication Satellite Conference, March (1990)

Pfleeger, S.H., Software Engineering: The Production of Quality Software, Macmillan, New York, New York (1987)

Shortlife, H.H., Buchanan, B.G., *"A Model of Inexact Reasoning in Medicine,"* Mathematical Biosciences, Vol.23 (1975)

Waterman, D.A., A Guide to Expert Systems, Addison-Wesley, Reading, Massachusetts (1986)

Image/Data Classification/Interpretation

FEATURE DETECTION IN SATELLITE IMAGES USING NEURAL NETWORK TECHNOLOGY

Marijke F. Augusteijn and Arturo S. Dimalanta
Computer Science Department
University of Colorado at Colorado Springs

ABSTRACT

This report describes a study of the feasibility of automated classification of satellite images[1]. Satellite images were characterized by the textures they contain. In particular, the detection of cloud textures was investigated. The method of second-order gray level statistics, using co-occurrence matrices, was applied to extract feature vectors from image segments. Neural network technology was employed to classify these feature vectors. The Cascade-Correlation architecture was successfully used as a classifier. The use of a Kohonen network was also investigated but this architecture could not reliably classify the feature vectors due to the complicated structure of the classification problem. The best results were obtained when data from different spectral bands were fused.

Keywords: Image Classification, Texture Analysis, Neural Networks.

INTRODUCTION

The extremely large volume of satellite image data that has been produced to date is difficult to classify for users. As an example, it has been estimated that only 5% of the Landsat images have ever been viewed by humans. Therefore, the ability to automatically classify satellite images is of keen interest to all potential users. If a computer could sort images by topic and possibly associate them with a level of interest (given some objective) then a human user would only have to search through a preselected set. This project is a feasibility study with the main purpose to determine if a specified feature can reliably be detected in a satellite image by computer.

An important task is to determine an appropriate set of features. Although it is sometimes important to detect actual objects in satellite images, most features are mainly visible as textures. For example, the waves in the ocean are observed as a texture, various forms of land (urban, agricultural or forests) appear as different textures, and the clouds in the sky form yet another texture. Thus, texture identification seems a valid means to classify images. This feasibility study will focus on the identification and discrimination of a single, possibly noisy texture. The feature selected is the texture of clouds. Clouds are particularly interesting because they do not necessarily cover an area. Clouds can be dense or sparse. When the clouds are sparse it will be possible to partially see through them and observe the surface below. In this case, the cloud texture will be intermixed with other textures. Thus, an automated technique for cloud identification must be capable of dealing with a considerable level of noise caused by these other textures.

Cloud detection and classification have been studied by many researchers (Goodman and Henderson-Sellers, 1988, and Rossow, 1989). Satellite observations of clouds have been utilized in atmospheric research ever since the first satel-

[1]This project was funded by CTA, Incorporated.

lite images were returned. Satellite images showing cloud formations are characterized by high variability of texture, irregularity of shapes, and a high level of boundary ambiguity, complicating cloud detection. Some researchers (Lee *et al.*, 1990), have gone beyond the identification task and have classified cloud textures as stratocumulus, cumulus, or cirrus. Accurate cloud detection is important for weather forecasting and the study of global changes in climate. In addition, there are other phenomena that produce cloudlike textures. For example, the smoke produced by a forest fire may look like a cloud. Also, the vapors released by volcanic eruption will be cloudlike in appearance. If clouds could be successfully identified even when mixed with other textures, it is expected that the same techniques will be applicable to the detection of large fires and volcanic activities.

Texture Identification

Texture identification has long been recognized as an important means for image classification, and many techniques to measure texture are available (Weszka *et al.*, 1979). A fairly simple procedure that has been successfully used by many researchers is *second-order gray level statistics* (Haralick *et al.*, 1973). This method is defined in the spatial domain and takes the statistical nature of the texture into account. A set of *co-occurrence matrices* is calculated, which measures the frequency of the simultaneous occurrence of two specified gray levels at two designated relative positions in an image segment (displaying the texture). Generally, four different matrices are used, each computing the frequency of gray level co-occurrence at neighboring positions in four different directions (horizontal, vertical, and along the two diagonal directions of the image).

A variety of measures can be employed to extract useful textural information from these matrices. Haralick *et al.* (1973) define fourteen different measures but consider four of them most useful. They are the angular second moment (sometimes called energy or homogeneity), the contrast, the correlation, and the entropy of a texture.

Neural Networks

Neural networks have recently become popular as general classifiers. For example, they were used in a cloud classification study (Lee *et al.*, 1990). The appeal of neural networks as pattern recognition systems is based upon several considerations. They appear to perform as well or better than other classification techniques and require no assumptions about the nature of the distribution of the pattern data. A comparison of neural networks to classical methods like K-nearest neighbor and discriminant analysis has shown that neural networks can achieve equal performance using a much smaller set of training data (Lee *et al.*, 1990). They have the capability to learn extremely complex patterns and are also suitable for multi-channel data fusion.

An important task is the selection of a neural network architecture appropriate for the application. Pattern recognition is often accomplished by means of a feedforward architecture. This type of network has its processing elements organized in different layers. The bottom layer accepts an input pattern and calculates the activations and outputs of its processing elements. The output values are then passed to the next layer, which performs a similar task. This continues until the top layer is reached. The output of the top layer represents the classification of the given pattern. The layers between top and bottom are often called hidden layers and are responsible for the

correct mapping between the input patterns and their classifications. The most familiar architecture in this class consists of three layers in which consecutive layers are completely connected, as shown in Figure 1.

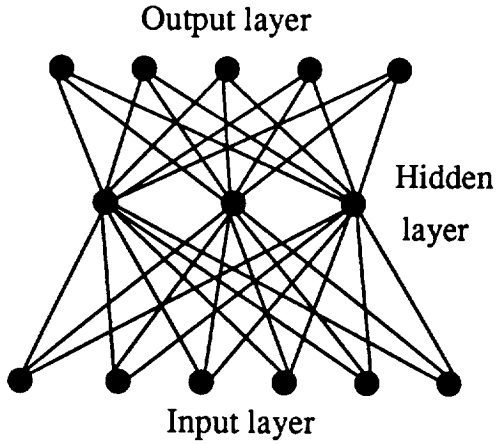


Figure 1. Processing elements and connections organized as a three layer neural network

The correct mapping is acquired during a training phase. In supervised training, the input patterns and the associated desired outputs are presented to the network. The network will update the connection strength between the processing units based on the difference between the desired and current outputs (the current measure of error). The most well-known updating scheme is backpropagation, which calculates an error measure at the output nodes and distributes this error back to the hidden nodes (Rumelhart *et al.*, 1986). However, although backpropagation has been used in numerous successful applications, it has several disadvantages. This learning method is extremely slow. The patterns that form the training set need to be presented many times, often thousands of times, before the network convergences to a solution. Sometimes, the correct solution will not be found. Although the algorithm attempts to find a global minimum of the total error, it may get trapped in a local minimum from which it cannot escape. Also, correct execution depends on the assignment of an appropriate number of nodes to the hidden layer(s). However, determining this number is more an art than a science. Many researchers have attempted to improve on backpropagation. One of these more recent architectures (Cascade-Correlation) is used in this study.

The Satellite Images

The set of satellite images used in this research consists of five scenes in both visible and IR spectral bands. They were obtained by an Advance Very High Resolution Radiometer (AVHRR) instrument. Images were available in five spectral bands. The wavelengths of each band are shown in Table 1.

The five scenes were obtained from the Great Lake area of the United States, the Atlantic Ocean, Barrow, Siberia and the Polar Cap. These scenes contain a variety of surface types, including clouds, water, sea ice, and land. Three of them show appreciable cloud cover with large variations in density. In areas containing sparse clouds, the underlying surface is clearly visible. Different types of surfaces appear through the cloud cover. Especially the Polar Cap scene, showing clouds against a background of ice, appears a challenging classification problem even for humans.

Table 1. Satellite Sensor Wavelength (μ m)

Satellite Band	Wavelength
Channel 1	0.58 - 0.68
2	0.725 - 1.1
3	3.55 - 3.93
4	10.5 - 11.3
5	11.5 - 12.5

ARCHITECTURES FOR TEXTURE ANALYSIS

A successful architecture developed to improve the slow learning characteristics of backpropagation is Cascade-Correlation (Fahlman and Lebiere, 1990). Like backpropagation, it incorporates supervised learning and has proved to be a powerful classifier. However, supervised learning generally does not reveal the underlying structure of the classification problem. In the simplest case, the various patterns will form distinct clusters with each cluster corresponding to a different class. However, it may happen that the clusters overlap. Then, the patterns belonging to the different classes are not well separated presenting a challenging problem to the classifier. In this case, a supervised architecture will experience more difficulty in learning the classification (and may even fail) but it will not show how the different classes relate. A self-organizing network like the one designed by Kohonen (1988) will show this underlying structure. This architecture employs unsupervised learning and organizes its units to reflect the relative configuration of the patterns.

The Cascade-Correlation Architecture

The Cascade-Correlation (Cascor) network is a dynamic architecture that incrementally builds its internal structure during training. Thus, the programmer need not be concerned with the appropriate number of units in the hidden layer(s) because the network itself will allocate the number of nodes required to solve the problem. The essence of the architecture is the following. Training in Cascor begins with the consideration of only two layers (input and output). They are fully connected and these connections are trained until no significant changes occur anymore. If, at that point, the total error is still unacceptably high, a hidden node will be positioned between these layers. The input connections to the new node are trained first. The algorithm attempts to maximize the correlation between the new node's activation and the output error of the network so that the new node may make up for the residual error to the greatest possible extent. The output connections are then trained by means of the quickprop algorithm, a second-order improvement to backpropagation (Fahlman, 1988). Hidden nodes are added, each one in its own separate layer, until the total error is below a preset threshold. Each hidden node is connected with all previously assigned hidden nodes, as well as with all input nodes, and is trained in isolation. Once trained, its input connections are frozen. Each hidden node is also connected with all output units. All output connections are trained after each addition of a hidden node. The basic architecture is shown in Figure 2. The resulting network is fast and capable of reliable classification.

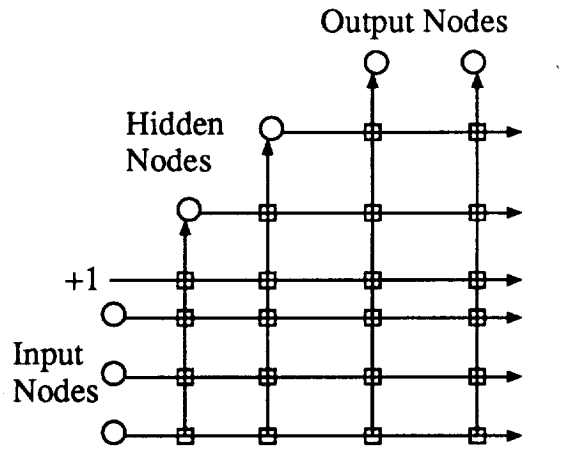


Figure 2. The Cascade Architecture
The vertical lines sum all incoming activations.

The initialization of the connection strengths is performed randomly between certain preset bounds. Thus,

when Cascor is run several times on the same data set, a different number of hidden nodes may be generated. These different runs are referred to as trials. Different trials, although trained on the same data set, may show different performance when used to classify the test data.

The Kohonen Self-Organizing Map

The Kohonen self-organizing map facilitates a better understanding of the underlying structure of the classification problem. This method provides a means to project a high dimensional vector space onto a lower (usually two) dimensional space which is simple to represent graphically. It creates a topology preserving map in which units that are physically located next to each other will respond to input patterns that are likewise next to each other.

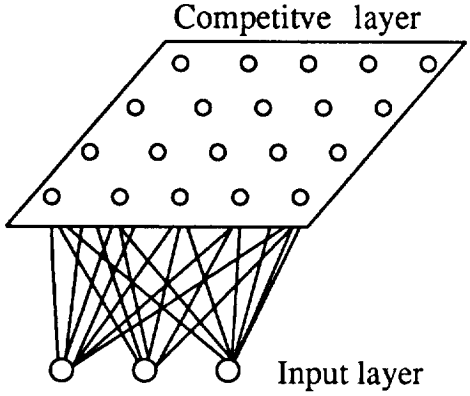


Figure 3. The architecture of the Kohonen Self-Organizing Map

The architecture consists of an input layer that is the size of the input pattern. This layer is completely connected to a (generally) two-dimensional organization of units as shown in Figure 3. The units in this second layer are competitive; that is, each one calculates an activation based on the input pattern and then enters into a competition with the other units in that layer. Each unit also represents a pattern, stored as the strengths (weights) of the connections leading to that unit from the input layer. The activation calculated by each one is proportional to the similarity between the input pattern and its stored pattern. The unit with the highest activation (whose stored pattern best approximates the current input) wins the competition. The winning unit as well as the units in its immediate neighborhood are selected for learning; that is, their weights are adjusted.

The architecture is initialized by assigning random weights (within certain preset limits) to all connections. Initially, it will be random which unit wins the competition. The winner and its neighbors will have their weights updated. The change is such that all weights move over a short distance towards the current input pattern which they begin to encode. Each presentation of an input pattern will move the weights of a set of units in the direction of that pattern. As training proceeds, the neighborhood affected will shrink. Thus, in the beginning a large group of units will be pulled towards a particular pattern while towards the end only a few will be moved. Eventually, after generally thousands of pattern presentations, the result of this kind of training is a topological organization of the units so that the ones encoding similar patterns will be geometrically grouped together. In this manner, the underlying structure of the clusters will become visible.

THE CLASSIFICATION EXPERIMENTS

The set of satellite images used in the experiments consists of 23 photographs showing 5 different scenes. Their distribution over the spectral bands is as follows. Bands 2 through 4 each contain 5 images (one of each scene), and bands 1 and 5 each have 4 images (with

the Great Lakes scene missing). All photographs are of size 512 x 512, contain 256 gray levels and have a resolution of 1100 meters per pixel. The images in bands 1 and 2 look most natural to the human eye since the corresponding wavelengths are in the visible or nearinfrared range, as shown by Table 1. The ones in bands 3 to 5 appear slightly unfamiliar since these are infrared photographs.

Several classification experiments were performed. All of them employed the same set of segments extracted from the images. All segments were selected using the Channel 2 photographs and had a size of 25 by 25 pixels. These segments were classified depending on the prevalent cloud pattern present. Not all cloud patterns appear the same. As mentioned before, a major cause for the differences in these patterns is cloud density. Assigning all the different densities to a single class did not seem reasonable. It was decided to define three classes of cloud patterns in the following way. When a segment is completely filled by clouds it will be labeled as *dense clouds*. Different patterns of dense clouds do occur, but these will all be assigned to the same class. When the cloud density is such that clouds fill the segment for at least two thirds of the area, this segment will be labelled as *medium clouds*. Finally, a segment showing light cloud cover such that less than one third of the area is actually covered by clouds is labelled as *sparse clouds*. All segments were selected to show as uniform a cloud pattern as possible. They do not cross texture boundaries, showing dense clouds in one part and possibly no clouds in another part. Thus, the medium and sparse cloud segments show clouds interspersed with land, water, ice, or a combination of these surfaces. All segments without any cloud cover are labelled as *no clouds*. These segments are filled with land, water, and ice, in various combinations.

Once the segments were selected in Channel 2, corresponding segments with the same coordinates were extracted from all other channels of the same scene. A feature vector was then formed for each segment in the following way. A set of four directional co-occurrence matrices was calculated for each one. The four prevalent measures, angular second momentum, contrast, correlation, and entropy were computed from each matrix. In order to measure a rotationally invariant texture, the feature values derived from the four directional matrices were averaged. The four values thus obtained were combined with the average gray level (which had to be scaled) and the standard deviation of the gray levels in each segment. The resulting six-dimensional feature vector was then normalized.

Classification with Cascade-Correlation

Cascor was used in three different experiments. In the first one, the feature vectors used for training the network and those that test the net were all taken from the same image. Thus, this experiment consisted of 23 independent tests, one for each of the 23 images. These tests were performed to get an initial impression of the classification capabilities but were not considered to be of major importance. The second experiment combined all images of a particular channel. It consisted of 5 independent tests, one for each channel. The third experiment combined information from different channels. Out of the many possible combinations five were selected that appeared most promising.

Experiment 1: Classification within a single image

All feature vectors generated from a single image were collected. In most cases, the image contained all four classes and provided 32 vectors, 8 for each class. One vector out of each group of 8 was randomly chosen as the test case. All others were used for training. All training vectors were randomly ordered so that the network would be exposed to all four classes simultaneously. Cascor always converged to a solution in a short time interval. The number of hidden nodes allocated varied from 3 to 11 with an average of about 6 when all 4 classes were present in the image. Images containing fewer classes generated fewer nodes. Each test consisted of a single trial. Classification in these tests scored over 90% on the average.

Experiment 2: Classification within a single channel

This experiment involved all feature vectors generated from images belonging to the same channel. These vectors were partitioned in a training set and in a test set. Four tests were performed in each channel. Each test used a different set of test items. Test items were obtained by random selection from each class and each image of a channel. The remaining vectors were used for training. A typical training set consisted of about 100 vectors, and about 16 vectors were used for testing. (The test and training sets in Channels 1 and 5 were somewhat smaller because one of the scenes was not available in these channels.)

Cascor was run five times on each training set. It always converged to a solution with a varying number of hidden nodes. Each group of five trials that were tested on the same data forms a test case. The test cases were labeled 1 through 4. It was observed that performance within a test case could vary considerably. This may be caused by the relatively small set of test data. Table 2 lists the average percentage of misclassifications of each test case and the misclassification percentage of the trial in each case that performed best. This table also shows the average number of hidden nodes generated during training and the overall average percentage of misclassifications observed in the channel. It is seen that the misclassification percentages are rather high and increase in the infrared channels.

More precise classification data can be obtained if the nature of the misclassification is taken into account. Three of the four classes correspond to different levels of cloud cover and are therefore quite similar. It may be considered less serious if a vector is misclassified within the group of cloud cover classes than when clouds are not recognized at all. Thus, it may also be important to make a distinction between segments showing some level of cloud cover and those containing no clouds at all. Tables 3 and 4 provide examples of the nature of the misclassifications obtained from the set of "best trials" of each test in Channels 2 and 3. These tables show the actual classifications horizontally and the classifications assigned by Cascor vertically. The numbers indicate fractions. Thus, the numbers along the diagonals indicate the fraction of correct classifications by the network, and the numbers off the diagonals show the fraction of misclassifications in each category.

Table 3 shows that most mis-classifications in Channel 2 were made between the different cloud cover categories. There are relatively few cases where a segment showing some cloud cover was taken for a segment that contained no clouds at all, or conversely.

Table 2. The number of hidden nodes and percentage of misclassifications in each test performed in each of the five channels

Channel	Test ID	Average Number of Hidden Nodes	Average Number of Misclassifications in %	Average in Channel	Misclassifications in best trial in %
1	1	21	25		17
	2	21	28		17
	3	22	22		17
	4	23	17	23	6
2	1	28	22		12.5
	2	25	28		25
	3	25	25		19
	4	26	22	24	12.5
3	1	28	40		31
	2	28	32		19
	3	30	31		12.5
	4	29	40	36	31
4	1	29	45		31
	2	29	46		37
	3	29	34		25
	4	30	45	42	25
5	1	23	42		33
	2	25	28		17
	3	24	52		50
	4	24	32	38	25

Table 3. The classification in Channel 2 of the best trials given as fractions

classification assigned by Cascor ↓	actual classification → dense clouds	medium clouds	sparse clouds	no clouds
dense clouds	0.67			0.04
medium clouds	0.17	1.00	0.13	
sparse clouds	0.08		0.81	0.04
no clouds	0.08		0.06	0.92

Table 4. The classification in Channel 3 of the best trials given as fractions

classification assigned by Cascor ↓	actual classification → dense clouds	medium clouds	sparse clouds	no clouds
dense clouds	0.50	0.17	0.06	
medium clouds	0.33	0.75	0.19	
sparse clouds	0.17	0.08	0.75	0.08
no clouds				0.92

When judging this result it should be taken into account that many of the *no clouds* segments show ice cover which, at least to the human eye, appears similar to a dense cloud cover. However, the neural network generally had no problem distinguishing between these similar textures. Table 4 shows the results in Channel 3. These are particularly interesting because all cloud segments in this channel were classified as containing some cloud cover. The classification in terms of *clouds* or *no clouds* was generally found to be above 80% in all channels.

Experiment 3: Classification using fused channel data

The previous experiment showed that the classification results differed for the various channels. Also, the kind of misclassifications seemed to vary slightly between the channels. In particular, in Channel 3 all clouds were classified as clouds, although misclassifications occurred between the different types. On the other hand, the detailed classification as different types of cloud cover in Channels 1 and 2 surpassed that of Channel 3. If information obtained from different channels were to be combined, better classification results could be expected. It was decided not to investigate all possible combinations but to select the more promising ones.

Channels 1 and 2 show the fewest misclassifications. Therefore, the data of these two channels were combined in an expectation of improved classification. Channel 3 is of interest because of its ability to distinguish between segments containing some cloud cover and those containing no clouds at all. The data of this channel were combined with those of Channel 2. Also, in order to make optimal use of the available data, it was decided to combine all five channels. Initially, these three types of test were performed. After it was observed that the Channel 2, 3 combination led to significantly improved results it was decided to also investigate the combined data of Channels 1, 2, 3 and Channels 2, 3, 4.

The channel data were combined by means of concatenating the appropriate feature vectors. Each feature vector used in the previous experiments has six components. As an example of how vectors were combined, consider the two sets used for the classification tests in Channels 1 and 2. Each vector in the Channel 1 set is generated from a specific segment in a Channel 1 image. Each one has a corresponding vector in the Channel 2 set generated from the analogous segment of the same scene in Channel 2. The information in the two channels was combined through concatenating each pair of corresponding vectors. Thus, the training set used for the Channel 1 and 2 combined test was of the same size as the training set used for testing Channel 1. However, each vector in the combined test had twelve components. The feature vectors of Channels 2 and 3 were combined in the same manner. The feature vectors of the combined Channels 1, 2 and 3 and Channels 2, 3 and 4 tests each had eighteen components. Finally, the corresponding feature vectors of all five channels were combined to form a thirty component vector for the classification experiment combining all channels.

The tests were conducted in the same manner as in the second experiment. Again, the feature vectors were partitioned into a training and a test set in four different ways. For each training set, Cascor was trained in five separate trials. All of them converged and were tested. Table 5 shows the results for the fused data. The nature of the misclassification of the "best trials" in each test is shown in

Table 5. The number of hidden nodes and percentage of misclassifications in each test case performed in the combined channels

Channels	Test ID	Average Number of Hidden Nodes	Average Number of Misclassifications in %	Average in Channels	Misclassifications in best trial in %
1 and 2	1	17	23		17
	2	18	10		0
	3	19	33		17
	4	17	32	24	17
2 and 3	1	19	9		0
	2	19	7		6
	3	18	15		0
	4	17	5	9	0
1, 2 and 3	1	12	12		0
	2	14	18		17
	3	13	10		8
	4	12	29	16	25
2, 3 and 4	1	17	15		0
	2	17	7		0
	3	18	9		0
	4	19	4	9	0
1, 2, 3, 4, and 5	1	12	15		0
	2	12	8		8
	3	12	10		8
	4	10	29	15	25

Table 6. The classification in Channels 2 and 3 combined of the best trials

classification assigned by Cascor ↓ \ actual classification →	dense clouds	medium clouds	sparse clouds	no clouds
dense clouds	0.92			
medium clouds	0.08	1.00		
sparse clouds			1.00	
no clouds				1.00

Table 7. The classification of all five channels combined of the best trials

classification assigned by Cascor ↓ \ actual classification →	dense clouds	medium clouds	sparse clouds	no clouds
dense clouds	0.88		0.08	
medium clouds	0.12	0.75		
sparse clouds		0.25	0.92	0.05
no clouds				0.95

Tables 6 and 7 for Channels 2 and 3 and all five channels combined, respectively.

Comparing the classification results of the fused Channel 1 and 2 data with the classification in Channels 1 and 2 separately, it is seen that the combined result gives the same overall classification performance. However, the fused data of Channel 2 and 3 showed significant improvement. Correct classification reached over 90% and matched the performance of the single image tests. In particular, the precise misclassification results displayed in Table 6 show that the "best trials" in each test had almost no misclassifications at all. When all five channels were combined, the classification performance dropped somewhat but is still better than classification in each of the channels separately. In particular, Table 7 shows that the separation between segments containing some level of cloud cover and those containing no clouds at all is quite good for these tests. The experiments in the three channel combinations showed similar results. It is remarkable, though, that in the Channel 2, 3 and 4 combination, there always was at least one trial that showed perfect classification.

Kohonen's self-organizing maps

The various sets of feature vectors were also used to produce the topological selforganizing maps. A topological map was generated for each channel separately as well as for the channel combinations discussed before.

When producing a map showing the organization of the feature vectors within a channel, the input layer must consist of six units since the single channel feature vectors have six components. In order to show the results of the combined channels, this input layer needs to be enlarged according to the increased size of the feature vectors. The competitive layer had 100 units organized as a 10 by 10 grid. The total number of feature vector presentations was 100,000. Convergence to a stable configuration was achieved. The size of the initial neighborhood was 5 by 5 and the initial learning rate was 0.2.

Figure 4 shows the topological map obtained from the Channel 1 data as an example. It is seen that the *no clouds* vectors are spread out most and show up in almost any segment of the plane. This is to be expected because these vectors represent many different textures. However, the different cloud types do not cluster very well either. Some clusters can be distinguished; for example, there is a *dense clouds* cluster consisting of five units in the top left quadrant of the plane. But some smaller clusters and isolated units representing the *dense clouds* texture are found in other locations. The *medium clouds* and *sparse clouds* patterns are distributed too. Similar distributions were observed in the other channels. It may be concluded that none of the five channels show strong clustering of feature vectors belonging to any of the four classes distinguished in the experiments. Thus, many of the feature vectors belonging to the same class are quite dissimilar. The clustering patterns of the larger vectors combining the results of more than a single channel were not significantly different.

CONCLUSIONS

The project researched the possibility of automated discrimination of a specified texture in AVHRR satellite images. The texture of cloud formations was selected and three different classes were defined based on the cloud density. Only a small set of satellite images was avail-

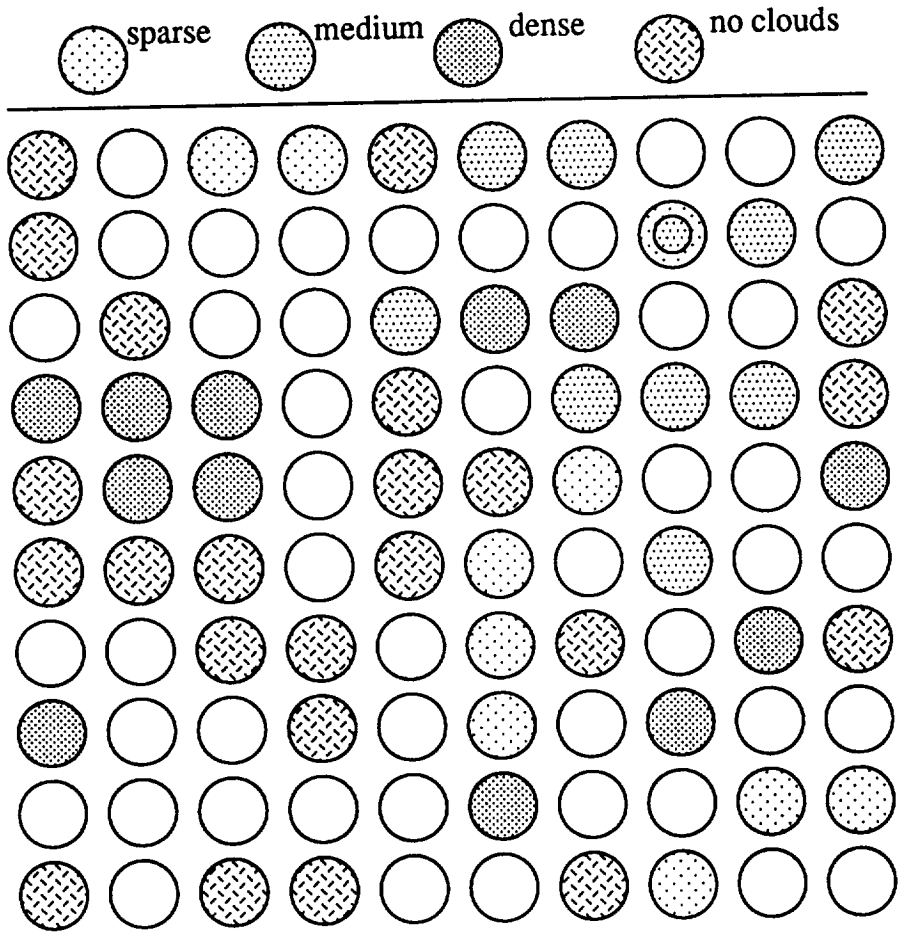


Figure 4. The clustering of the dense, medium, and no clouds feature vectors in Channel 1

able. Taking the difficulty of the classification into account, it may be concluded that this project was successful in the sense that it was found possible to discriminate cloud textures from all other textures with reasonable accuracy. Segments showing the various levels of cloud cover were extracted. In many segments, the cloud textures were mixed with various levels of noise due to small gaps in the cloud cover. The method of second-order gray level statistics was used to obtain feature vectors from these segments. The clustering properties of these vectors was studied by means of the Kohonen self-organizing maps.

The vectors generated by the *no clouds* class did not cluster very well as should be expected. These vectors represent many different textures and will show large variability. It should also be anticipated that the *sparse clouds* vectors would not cluster well. This turned out to be generally the case (although the largest cluster observed in any of the maps belonged to the *sparse clouds* class). The *medium clouds* and *dense clouds* feature vectors were expected to cluster better as compared to the other two classes, but this was found not to be the case. In order to obtain better clustering properties of feature vectors, different preprocessing methods could be studied. Possible can-

didates are the two-dimensional Fast Fourier transform, the Gabor transform and wavelets expansions. However, it should be realized that cloud textures show large variability and the classification problem may be inherently difficult, independent of which preprocessing technique is used.

Given this large variability in feature vectors within a class, it does not seem advisable to use a self-organizing neural network architecture for classification. The topological maps generated by the Kohonen network were of interest because they revealed the complexity of the classification problem. However, if this architecture had been used as a classifier, it would have generated many misclassifications. A neural network architecture employing supervised learning is better suited for this type of classification as demonstrated by this project. The Cascade-Correlation network performed well. The best results were obtained when data from Channels 2 and 3 or Channels 2, 3 and 4 were fused. In these cases, the four classes could be distinguished with an average accuracy of 91%. Moreover, several tests in these channel combinations showed no misclassification at all. If these better performing trained networks could be recognized in advance, much better classification results could be obtained.

We recently became aware of a similar study performed by Slawinski *et al.*, 1991. These researchers used the backpropagation architecture to classify different levels of cloud cover against an ocean background in AVHRR images. They used the pixel gray levels of small image segments together with first-order statistics measures as inputs to the neural networks. Their best results (93% correct classification) are similar to the best classifications obtained in our project. However, the ocean provides a rather homogeneous background and the variability in their images is essentially introduced by the cloud textures. When the background itself shows large variability, as in the majority of the images used for our project, classification methods that are largely based on the actual values of the pixels may not be successful.

This feasibility study has proved the possibility of automated satellite image classification. Future research could focus on the distinction of many different textures in these images. Eventually, a software package could be implemented that partitions an image into a set of overlapping segments and then scans each segment in an attempt to classify it according to its dominant texture. A set of identified textures could be used as an index in a data base through which images could be stored and retrieved. Research will be required to specify an appropriate set of textures. Various preprocesing techniques need to be investigated with respect to the clustering properties of the generated feature vectors. Additional research may be required to select the most appropriate neural network architecture. Based on the current results, Cascade-Correlation seems a good candidate for the expanded classification task. However, there is some evidence that the generalization characteristics of Cascade-Correlation are not as good as those of the backpropagation network (Crowder, 1990). Thus, it may be useful to consider additional architectures.

Acknowledgements: The satellite images were made available by CTA, Incorporated. The code for the Cascade-Correlation neural network was made available by R. Scott Crowder, III of Carnegie Mellon University. The code for the Kohonen self-organizing maps was written by Juan C. Soto, Jr. of the Univer-

sity of Colorado at Colorado Springs. The training of the Kohonen network was performed by Tammy L. Skufca, also of the University of Colorado at Colorado Springs.

REFERENCES

Crowder, R. S. (1990). Predicting the Mackey-Glass Timeseries with Cascade-Correlation Learning. In D. S. Touretzky (Ed.), *Proceedings of the 1990 Connectionist Models Summer School* (pp. 117-123). San Mateo: Morgan Kaufmann.

Fahlman, S. E., and Lebiere, C. (1990). The Cascade-Correlation Learning Architecture. In D. Touretzky (Ed.), *Advances in Neural Information Processing Sytems 2* (pp. 524-532). San Mateo: Morgan Kaufmann.

Fahlman, S. E. (1988). Faster-Learning Variations on Back-Propagation: An Empirical Study. In D. Touretzky (Ed.), *Proceedings of the 1988 Connectionist Models Summer School.* San Mateo: Morgan Kaufmann.

Goodman, A. H., and Henderson-Sellers, A. (1988). Cloud detection and analysis: A review of recent progress. *Atmos. Res.*, 21, 203-228.

Haralick, R. M., Shanmugam, K., and Dinstein, I. (1973). Textural Features for Image Classification. *IEEE Transactions on Systems, Man, and Cybernetics*, 3, 610-621.

Kohonen, T. (1988). The Neural Phonetic Typewriter. *Computer*, 21, 11-22.

Lee, J., Weger, R. C., Sengupta, S. K., and Welch, R. M. (1990). A Neural Network Approach to Cloud Classification. *IEEE Transactions on Geoscience and Remote Sensing*, 5, 846-855.

Rossow, W. B. (1989). Measuring cloud properties from space: A review. *J. Climate*, 2, 201-213.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning Internal Representations by Error Propagation In D. E. Rumelhart and J. L. McClelland (Eds), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1: Foundations (pp. 318-364). Cambridge, Mass.: MIT Press.

Slawinski, O., Kowalski, J., and Cornillon, P. (1991). A Neural Network Approach to Cloud Detection in AVHRR Images, *Proceedings of the International Joint Conference on Neural Networks, Vol. I*, 283-288.

Weszka, J. S., Dyer, C. R., and Rosenfeld, A. (1979). A Comparative Study of Texture Measures for Terraine Classification. *IEEE Transactions on Systems, Man, and Cybernetics*, 6, 269.

Design of Neural Networks for Classification of Remotely Sensed Imagery

Samir R. Chettri
Hughes-STX at NASA/Goddard Space Flight Center
Greenbelt, MD 20771

Robert F. Cromp
Code 934
NASA/Goddard Space Flight Center
Greenbelt, MD 20771

Mark Birmingham
Princeton University, New Jersey

Abstract

As currently planned, future Earth remote sensing platforms (i.e., Earth Observing System [EOS]) will be capable of generating data at a rate of over fifty Megabits per second. To address this issue the Intelligent Data Management (IDM) project at NASA/GSFC has prototyped an Intelligent Information Fusion System (IIFS) that uses backpropagation neural networks for the classification of remotely sensed imagery. This is part of the IDM strategy of providing archived data to a researcher through a variety of discipline–specific indices.

In this paper we discuss classification accuracies of a backpropagation neural network and compare it with a maximum likelihood classifier (MLC) with multivariate normal class models. We have found that, because of its nonparametric nature, the neural network outperforms the MLC in this area. In addition, we discuss techniques for constructing optimal neural nets on parallel hardware like the MasPar MP-1 currently at NASA/GSFC. Other important discussions are centered around training and classification times of the two methods, and sensitivity to the training data. Finally we discuss future work in the area of classification and neural nets.

1 Introduction

With the expected explosive growth of data generated by Earth orbiting platforms such as the Earth Observing System (EOS), it is imperative that the data be rapidly archived and made available to the researcher through a variety of discipline–specific indices. To address this issue, the Intelligent Data Management (IDM) project at NASA/GSFC has prototyped an Intelligent Information Fusion System (IIFS) that classifies satellite data from a number of spectral bands into a number of land use/land cover categories [Anderson 76] and provides rapid access to the classified data as well as the raw data. The choice of land use/land cover categories is part of a larger plan to classify images based on a scientist's specific research interests.

Management of the EOS data can be considered as two overlapping problems: characterization of the data content and subsequent archiving of images; and efficient querying of the resulting voluminous database. The first problem can be solved independently of the choice of database technology, and is elaborated upon in this paper.

In this paper we discuss the use of neural nets for the classification of remotely sensed imagery. In particular we compare backpropagation neural nets (BPNN) with a Gaussian maximum likelihood classifier (GMLC). Some of the items we compare include training time, classification time, accuracy of classification, and sensitivity of classification accuracy to the training set. This study will thus help researchers decide on what classification method to apply, given the constraints of their problem.

The body of the paper is divided into three sections. First, we briefly discuss the algorithms for the neural net and the MLC methods. Next, we compare and contrast the two methods. Finally, we discuss selection criteria for each of the two methods and conclude with our future research directions.

2 Neural net and maximum likelihood classification algorithms

In this section we discuss the basic algorithms for training and classification for the neural net (NN) and Gaussian maximum likelihood classifiers (GMLC). For more details on both topics, refer to [Andrews 72] and [Hertz 91].

2.1 Maximum likelihood classification

The job of designing the pattern classifier consists of first dividing the feature space into decision regions and then constructing a classifier so that it will identify any measurement vector $\mathbf{X}$ as belonging to the class corresponding to the decision region in which it falls.

The maximum likelihood decision rule allows us to construct discriminant functions for the purposes of pattern classification [Andrews 72]. Given K classes, let $f(\mathbf{X} \mid S_k)$ be the probability density function (pdf) associated with the measurement vector $\mathbf{X}$, given that $\mathbf{X}$ is from class k. Let $P(S_k)$ be the *a priori* probability of class k. We can use the **maximum likelihood decision rule** to identify the class to which $\mathbf{X}$ belongs. It can be stated as follows:

$$\textit{Decide } \mathbf{X} \in S_k \textit{ iff } f(\mathbf{X} \mid S_k)P(S_k) \geq f(\mathbf{X} \mid S_j)P(S_j), j = 1, 2, \cdots K \ .$$

The products $f(\mathbf{X} \mid S_k)P(S_k)$, where $k = 1, 2, \cdots K$ correspond to discriminant functions $g_1(\mathbf{X})$, $g_2(\mathbf{X}), \cdots g_K(\mathbf{X})$. Thus these functions are evaluated at $\mathbf{X} = \mathbf{X}_i$ where $\mathbf{X}_i$ is the unknown vector; next, the maximum of these functions $g_k(\mathbf{X}_i)$ is determined and the unknown vector is assigned to the class k.

The discriminant function for the multivariate normal density can be written as

$$g_k(\mathbf{X}) = \ln[P(S_k)] - \frac{1}{2}\ln|\mathbf{\Sigma}_k| - \frac{1}{2}(\mathbf{X} - \mathbf{U}_k)^T \ \mathbf{\Sigma}_k^{-1} \ (\mathbf{X} - \mathbf{U}_k). \tag{1}$$

In the above equation, both Σ_k (the variance–covariance matrix) and $\mathbf{U}_k$ (mean vector) are provided by the user. In practice, training samples are used to obtain *estimates* of Σ_k and $\mathbf{U}_k$. Also from equation (1) we see that once the training statistics are generated, only the quadratic kernel varies with each input vector $\mathbf{X}$. Such a classifier is called a Gaussian Maximum Likelihood Classifier (GMLC) and is used in our classification experiments. It is a parametric supervised technique for estimation of *a posteriori* probabilities.

2.2 Backpropagation

The backpropagation algorithm is the backbone of much of the current resurgence of research into neural nets [Hertz 91]. With respect to pattern recognition, backpropagation can be considered to be a nonparametric technique for estimation of *a posteriori* probabilities [Wan 90].

The backpropagation network consists of a series of layers: an input layer, an output layer and one or more hidden layers. Each layer has a number of nodes or processing elements (PE). Each node in a layer is connected to every node in the next layer and the propagation of information is unidirectional. Also, in our simulations, connections are only permitted between nodes belonging to adjacent layers. Each connection has a value associated with it called its weight, and each PE has a value associated with it called a threshold value.

To find the output of any node we first sum the products of the output of all the nodes before it with the weights associated with each connection. Next we subtract the threshold value of the node from this sum, and finally we pass this value through an activation function that determines the output of the current node. The activation function used in this study is the sigmoid function defined as

$$f(h) = \frac{1}{1 + \exp^{-kh}} \tag{2}$$

where $h = \sum w_i \xi_i - \Theta$, w_i are the weights, ξ_i are the inputs to the current node (or output of nodes in the previous layer), and Θ is the threshold value in the the current PE.

The training phase of backpropagation gives a method of changing the weights in a network such that it learns a series of input/output pairs (ξ_k^μ, ζ_i^μ) where ξ_k^μ is the k^{th} input for the μ^{th} pattern, and ζ_i^μ is the correct output for the i^{th} output unit for the μ^{th} pattern. The basis for the weight change is gradient descent, thus the weights w_{jk}^r are changed by an amount Δw_{jk}^r that is proportional to the derivative error function E with respect to the weights, where w_{jk}^r is the weight that lies on a connection between the j^{th} PE of one layer with the k^{th} PE of the previous layer, and r indicates the number of the current layer. The most commonly used error function is the quadratic error function [Hertz 91] defined as

$$E = \frac{1}{2} \sum_{\mu i} [\zeta_i^\mu - O_i^\mu]^2 . \tag{3}$$

Here, the summation is over *all* training samples, and O_i^μ is the network output for a given input pattern for which the expected output is ζ_i^μ.

Training proceeds by randomly selecting the weights in the net, passing the input pattern through the network, getting the resulting output, obtaining $\Delta w_{jk}^r = -\eta \frac{\partial E}{\partial w_{jk}^r}$, and finally updating

Table 1: Distribution of data, Blackhills and DC data sets

Class	# of Pixels Blackhills		# of Pixels DC		Class name
	Training	Entire image	Training	Entire image	
0	453	6676	73	2668	Urban
1	478	42432	74	776	Agricultural
2	464	16727	75	3733	Rangeland
3	482	194868	75	13826	Forested Land
4	0	0	0	0	Water bodies
5	0	0	0	0	Wetland
6	368	1441	74	936	Barren
7	0	0	0	0	Tundra
8	0	0	0	0	Perennial snow and ice

the weights by using $w^r_{jk,\text{new}} = w^r_{jk,\text{old}} + \Delta w^r_{jk}$. Training is done either for a maximum number of iterations or until the error E goes below a pre–defined threshold level. At this point the trained network can be used on data in feed–forward mode for the purposes of classification.

3 Experimental Method

In this section we describe the data that we used to compare our neural net (NN) classifier and Gaussian Maximum Likelihood classifier (GMLC). In addition we discuss the selection process that we employed for the training and testing data. Finally, we describe the training and testing methodology used.

3.1 Description of data set

Two data sets were used for the purpose of comparing the the GMLC and the NN approach. The first is the Blackhills data set, taken from the Landsat 2 multispectral scanner (MSS) (see Figure 1). The spectral bands are 0.5 - 0.6μm (green), 0.6 - 0.7μm (red), 0.7 - 0.8μm (near–infrared) and 0.8 - 1.1μm (near–infrared). These bands correspond to channels 4 through 7 of the Landsat sensors. There are 262,144 pixels corresponding to a 512 $\times$ 512 image size, and each pixel represents 79m $\times$ 79m on the ground. The image region covers a range of latitudes from 44°15′ to 44°30′ and longitudes from 103°30′ to 103°45′; the images were obtained in September 1973. The ground truth was also provided in the form of United States Geological Survey level II land use/land cover data [Anderson 76]. Since we were only interested in level I classification, the different classes were conglomerated into the various higher level classes in the hierarchy; the distribution of pixels is shown in column three of Table 1.

The second data set has been used previously in [Campbell 89], which, to our knowledge, was the first open–literature publication of the use of the backpropagation network to do classification of remotely sensed imagery. In contrast to that publication, we will only be using the USGS level I land use/land cover scheme for classification. The first four spectral bands from a LANDSAT-4

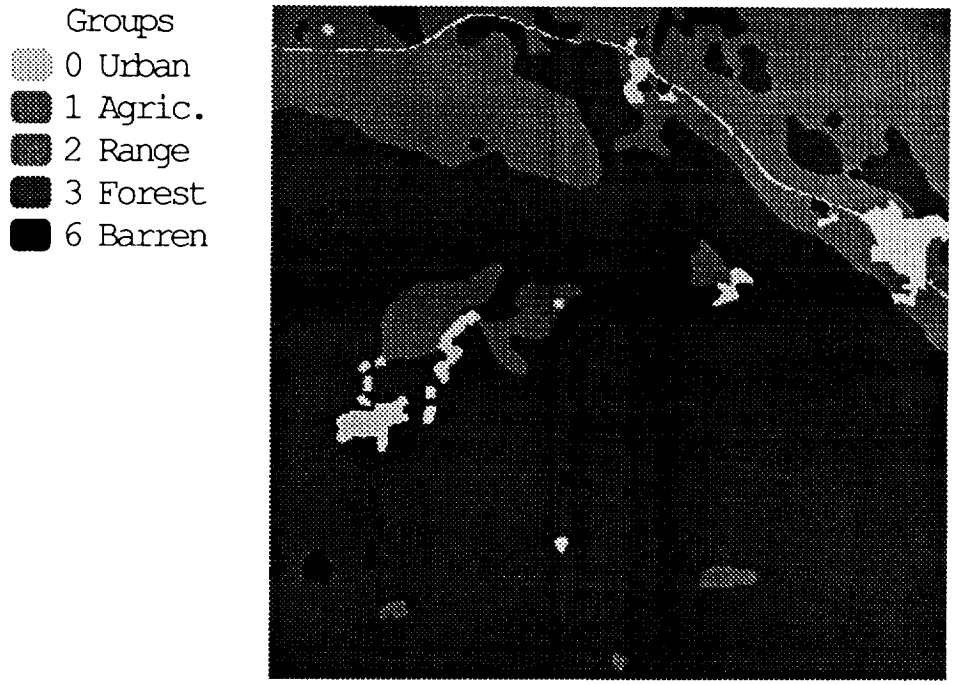


Figure 1: Ground truth for the Blackhills image

thematic mapper (TM) image were used, and the corresponding spectral bands were 0.45 - 0.52μm, 0.52 - 0.60μm, 0.60 - 0.69μm and 0.76 - 0.90μm respectively. There are a total of 22,801 pixels in a 151 $\times$ 151 grid, and each pixel is representative of a $30m \times 30m$ area on the ground. Only 21,939 pixels of valid ground truth were available, and these are tabulated in column five of Table 1. Figure 2 shows a gray-level thematic map, with the various classes labeled. The area covered is about 25 miles SSE of Washington, DC, and is called the DC data set.

3.2 Selection of training and test data

The most important point to note is that identical data were presented to the NN and GML classifiers for training and testing. The ground truth was viewed on a display device to get an idea of the spatial distribution of the ground truth pixels. According to [Richards 86], a minimum sample size of 60 pixels is necessary for accurate classification. Also, according to [Campbell 87], a large number of smaller training sites should be used rather than a few large ones. Following these recommendations, we formed training sets from both the TM (DC data set) and MSS (Blackhills data set) scenes. The results are summarized in columns two and four of Table 1.

3.3 Training and testing

The training set and the test set are disjoint. The classifiers were derived from the training group and the error estimate obtained from the test group. This method is known as the "holdout" or H method of estimating errors. The training data itself consists of a series of sites from each class in the image. For the GMLC we can compute the mean vectors and covariance matrices for each site separately and combine them to form the class mean vectors and covariance matrices. For the NN

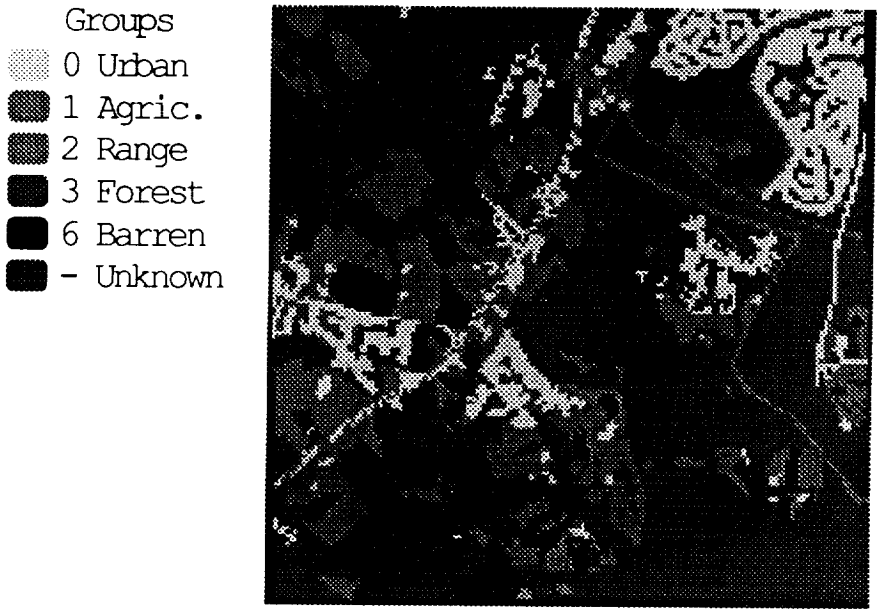


Figure 2: Ground truth for the DC image

approach, the site information is not important. However, the channel information, which is an integer in the range [0 255], is scaled to [0.1 0.9] for training. The training of the NN is achieved by repeatedly presenting the data to the net and performing the backpropagation algorithm as described in section 2.2. Training in the NN is completed when either the error as described in equation (3) goes below a threshold level, or a maximum number of iterations of the BP algorithm is reached. It is important to avoid overtraining the net as it would classify the training data perfectly but would not perform as well on the testing data.

An alternative method for training and testing data is recommended in [Weiss 89] and is called *leaving-one-out*. The principle is very simple and involves taking $n-1$ points from the sample and training the classifier on that information. The n^{th} point is then classified and this training and testing procedure is repeated for all n points in the set. Quite clearly, for our data it is entirely infeasible to use the leaving-one-out process since we will have to design $n = 262,144$ and $n = 22,801$ classifiers for the Blackhills and DC data sets respectively. However, with large sample sizes (which is the case for our data) the accuracy in estimating error is adequate by the H method [Kanal 68], hence our selection of that procedure.

The training of the NN proceeds on the basis that it is a function optimization procedure. Remembering that the function optimization process is sensitive to initial conditions, we:

1. randomize the initial selection of weights and thresholds;

2. randomize the order of the training data.

The effect of this is to produce different neural nets for each set of initial conditions. Thus, when we compare the NN and GMLC accuracies, we will be referring to the *average* correctly classified by the NN, while there will be only one value for the GMLC. Training of these multiple neural nets can be achieved on the MasPar MP-1, with each processing element generating an independent NN. The best net is one that obtains the lowest error on the test set, and it is selected for general use.

4 Comparing backpropagation (BP) with Gaussian maximum likelihood classification (GMLC)

In this section we compare BP with GMLC under different sets of categories. These categories include time for training, time for classification, memory requirements, and classification accuracy. Instead of exact calculations, we give order–of–magnitude estimates for these quantities. It is important to note that we assume that our neural net is restricted to one hidden layer, because according to Kolmogorov's theorem [Hecht-Nielsen 90], a three layer neural net can be constructed that performs any continuous mapping with $(2N + 1)$ PE's in the hidden layer, where N is the number of elements in the input layer. Another assumption is that the output layer has m nodes where m is proportional to N. Note that N is also the dimension of the unknown vector whose class we are trying to determine. This notation will be used in the subsequent subsections.

4.1 Training time

Training time in the neural net can be shown to be $\mathcal{O}(N^6)$ based on arguments in [Muhlenbein 90]. The training phase of GMLC has a worst case complexity of $\mathcal{O}(N^3)$.

While it seems quite clear the training time for the GMLC is significantly less, the *current, off–the–shelf* availability of hardware to do backpropagation training reduces the advantage of the GMLC. Since BP is a far more general process, hardware will continue to be supported and developed for it, whereas since the GMLC is a specific method of classification, it is uneconomical to develop custom hardware for this process. In addition, we have only discussed a simple BP scheme. In fact there exist a number of speed–up procedures [Hertz 91], that would make BP competitive with GMLC in software implementations.

Also, while the GMLC is suited for similar types of data (in our case spectral information), it is unsuited for multi–source data, since the underlying distribution may change when one adds (say) elevation data [Benediktsson 90]. The NN handles these problems in an effective manner. In addition, our application permits the training to be performed off–line, thus eliminating the time factor entirely.

4.2 Classification time

Both the NN and GMLC method can be shown to take constant time for the classification of one pixel. Again, the availability of hardware makes the NN method more attractive. In addition, even in software, the time needed for neural network computations can be considerably decreased by using integer calculations for the sigmoid function [Birmingham 91]. In this paper, a Taylor series

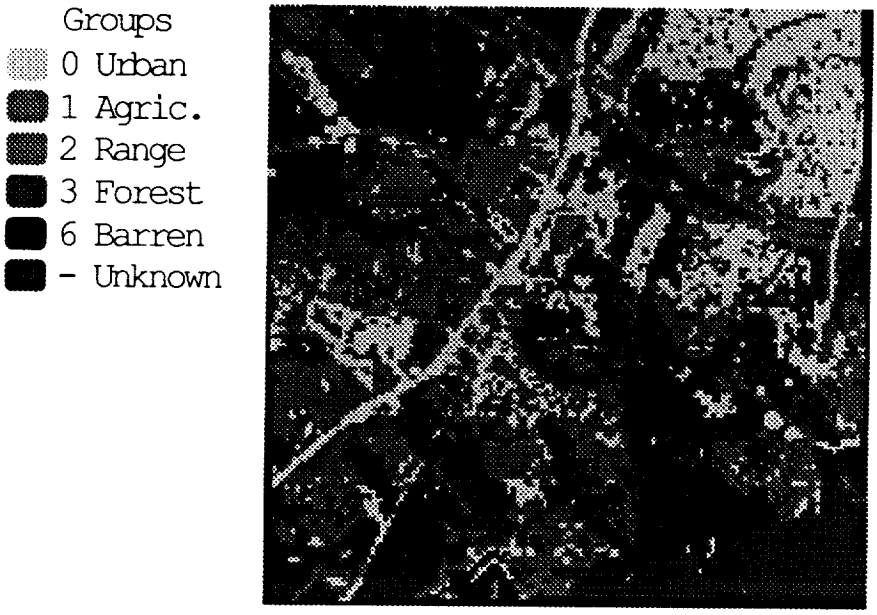


Figure 3: Maximum likelihood classified DC image

approximation to the sigmoid with only integer fractions is used. It was found that an almost six–fold speed–up factor can be obtained. Another advantage of the NN is that the same hardware can be used for training and feedforward classification, which is not the case for the GMLC.

4.3 Accuracy

The accuracy of each method can be summarized by the *contingency table*, which is an $R \times R$ matrix of numbers, where R is the number of classes. Each entry C_{ij} in the matrix represents the number of times a pixel in class i was put into class j. C_{ii} is the number of correct classifications in class i.

For the DC data set we have two sets of contingency tables. In Table 2, we present the GMLC accuracy results for the training and test data respectively. In Table 3, *typical* accuracy results for the NN are presented. In addition, Table 4 presents the average percent correctly classified (PCC), the maximum PCC, and the minimum PCC for all the nets that were trained. We see that even the minimum PCC for the NN exceeds the PCC value obtained by GMLC. The number of nets trained to get these readings was six. For the purposes of visual comparison, the classified images are shown in Figures 3 and 4.

We have two sets of contingency tables for the Blackhills data set. In Table 5, we present the GMLC accuracy results for the training and test data, respectively. In Table 6, *typical* accuracy results for the NN are presented. In addition, Table 7 presents the average PCC, the maximum PCC, the minimum PCC as well as the standard deviation of the PCC for all the nets that were

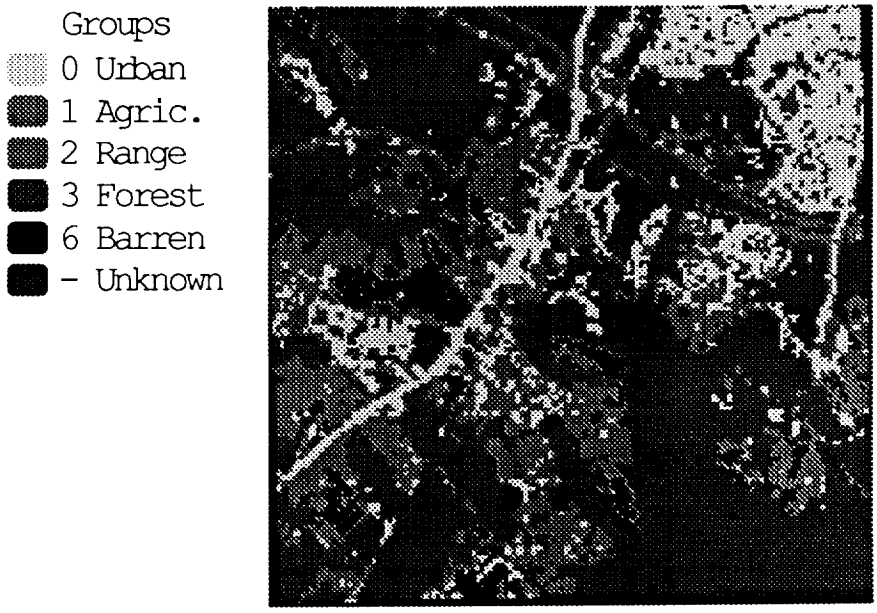


Figure 4: Neural net classified DC image

Table 2: Contingency table for GMLC, DC training data on left (PCC = 0.827), DC test data on right (PCC = 0.623)

	0	1	2	3	6	0	1	2	3	6
0	**68**	0	5	0	0	**1843**	219	505	16	12
1	2	**56**	12	4	0	30	**380**	158	14	120
2	6	5	**64**	0	0	605	1132	**1472**	220	229
3	0	0	1	**72**	2	1661	715	1634	**9408**	333
6	0	27	0	0	**47**	46	380	100	3	**333**

Table 3: Contingency table for NN, DC training data on left (PCC = 0.871), DC test data on right (PCC = 0.677)

	0	1	2	3	6	0	1	2	3	6
0	**68**	0	5	0	0	**1714**	94	751	26	10
1	1	**47**	15	0	11	15	**195**	243	21	228
2	2	1	**72**	0	0	440	404	**2054**	322	438
3	0	0	0	**75**	0	1144	133	1952	**10227**	295
6	0	13	0	0	**61**	33	251	170	4	**404**

Table 4: Statistics for NN performance on DC test data set

#	Av.	Max.	Min.	Std. dev.
6	0.675	0.688	0.665	0.093

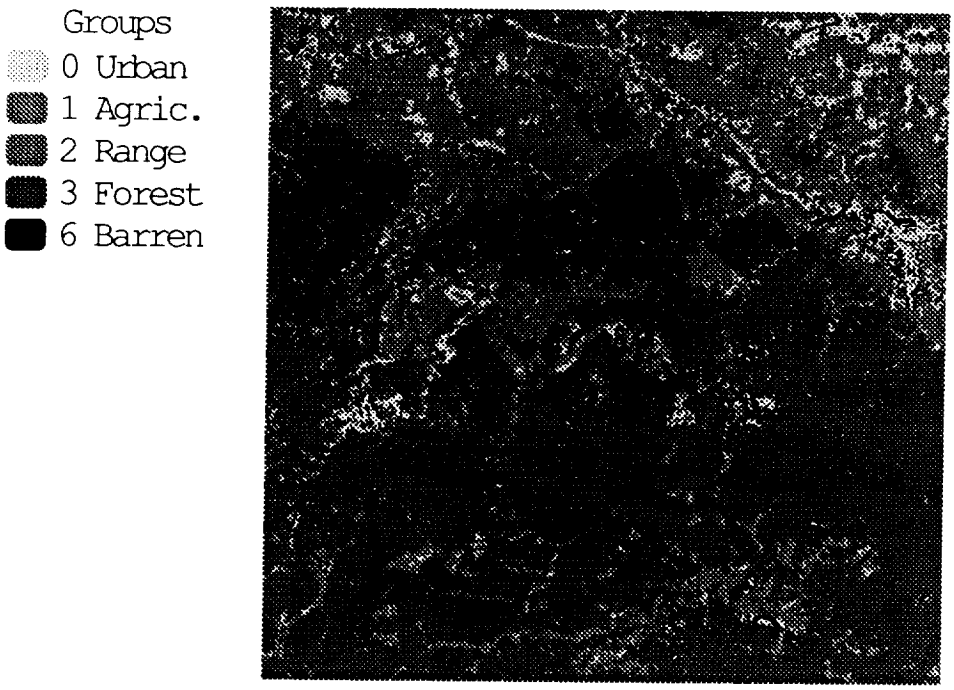


Figure 5: Maximum likelihood classified Blackhills image

trained. We see that even the minimum PCC for the NN exceeds the PCC value obtained by GMLC. To compare the classified images visually, refer to Figures 5 and 6.

It is important to note that the contingency table can be used as an aid to further improving classification accuracy. This is called the conditional probabilities matrix (CPM) technique and is described in detail in [Cromp 91]. Using this technique, a distance measure representing the error was reduced by approximately 50%. Of course, the method applies to the contingency tables produced by both the NN and GMLC.

4.4 Memory requirements

Both the NN and the GMLC can be shown to require $\mathcal{O}(N^2)$ memory elements.

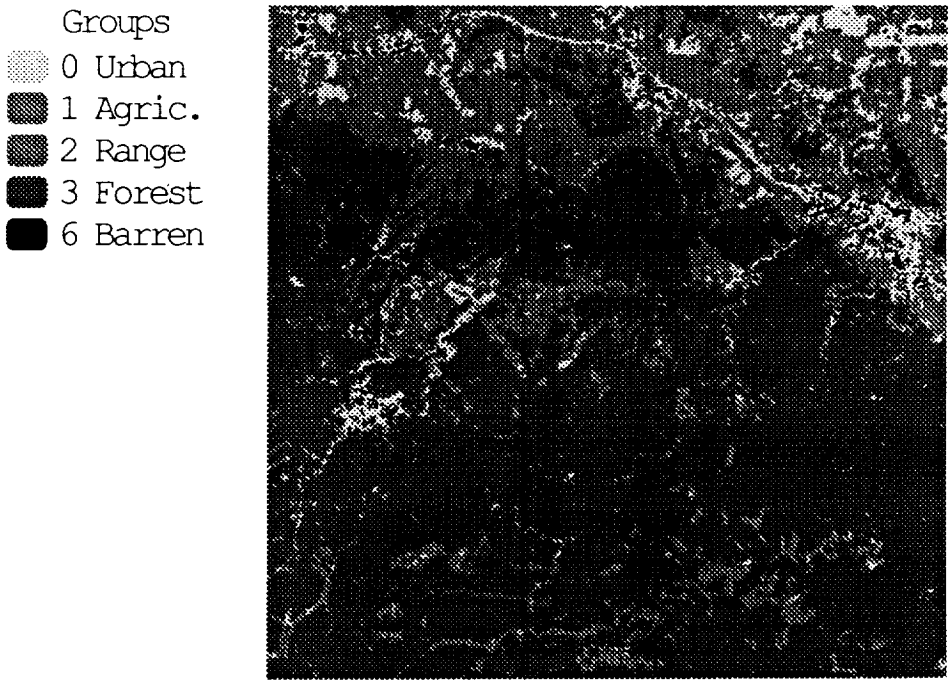


Figure 6: Neural net classified Blackhills image

Table 5: Contingency table for GMLC, Blackhills training data on left (PCC = 0.571); Blackhills test data on right (PCC = 0.653)

	0	**1**	**2**	**3**	**6**	**0**	**1**	**2**	**3**	**6**
0	**236**	72	91	18	36	**2425**	731	1307	876	884
1	26	**316**	135	0	1	6631	**16140**	15741	1463	1979
2	16	119	**279**	43	7	1840	3450	**9333**	1165	475
3	1	4	77	**385**	15	4077	8761	25804	**141644**	14100
6	61	28	78	136	**65**	157	116	147	442	**211**

Table 6: Contingency table for NN, Blackhills training data on left (PCC = 0.578); Blackhills test data on right (PCC = 0.727)

	0	**1**	**2**	**3**	**6**	**0**	**1**	**2**	**3**	**6**
0	**274**	74	86	16	3	**3021**	709	1413	892	188
1	26	**323**	124	5	0	7689	**16700**	15285	1998	282
2	21	115	**284**	44	0	2134	3610	**9360**	1128	31
3	5	4	91	**381**	1	2183	11572	20749	**159832**	50
6	87	31	75	139	**36**	228	125	155	473	**92**

Table 7: Statistics for NN performance on Hills test data set

#	Av.	Max.	Min.	Std. dev.
6	0.736	0.754	0.706	0.019

5 Concluding remarks and future work

In this research we have compared the backpropagation neural network (BPNN) with Gaussian maximum likelihood classification (GMLC). The accuracy level of BPNN (i.e., the number of correctly classified pixels in a test set) is better than the accuracy obtainable by GMLC. This is because the BPNN makes no *a priori* assumptions about the underlying densities of the data. The memory requirements and classification time were shown to be equivalent for both methods. Finally, the time for training was discussed. In this case, the GMLC takes less time than the BPNN; however, this is not considered to be a disadvantage because: the training can be performed off–line in our application; special purpose BPNN hardware exists for training and testing; and a variety of speed–up techniques are available for BP in software. From these results we feel that the BPNN is a better candidate for doing supervised characterization of remotely sensed data.

Recently, a new type of neural network called the probabilistic neural network (PNN) has been developed [Specht 90]. It uses the technique of Parzen windows for nonparametric density estimation and uses the technique of maximum likelihood estimation for classification. It offers the twin advantages of being available in hardware [Washburne 91] as well as being considerably quicker to train than BP. We will investigate the application of such classifiers to our problem. In addition we will research the use of ancillary data such as texture and spatial information to improve our classification accuracy.

We have mentioned the MasPar MP-1 as a parallel computer alternative in previous sections. It will be the focus of IDM to implement parallel code for the BPNN as well as the GMLC, thus providing fast alternatives to the remote sensing researcher.

6 Acknowledgements

A number of people within and without the IDM group have contributed to this work. The authors would like to thank William Campbell, Erik Dorfman, George Fekete, Co Horgan and Nicholas Short, Jr. for their input into the content of this paper.

References

[Anderson 76] J. R. Anderson, E. E. Hardy, J. T. Roach, and R. E. Witmer. A land use and land cover classification system for use with remote sensor data. Geological Survey Professional Paper 964, United States Government Printing Office, Washington, D.C., 1976.

[Andrews 72] H. C. Andrews. *Introduction to mathematical techniques in pattern recognition.* Wiley–Interscience, New York, 1972.

[Birmingham 91] M. Birmingham. Acceleration of neural networking through the use of integer approximations for floating point operations. IDM memo 13, NASA, Intelligent Data Management, Code 934, Greenbelt, Maryland 20771, 1991.

[Benediktsson 90] J. A. Benediktsson, P. H. Swain, and O. K. Ersoy. Neural network approaches versus statistical methods in classification of multisource remote sensing data. *IEEE Trans. on Geoscience and Remote Sensing*, 28(4):540–551, 1990.

[Campbell 87] J. B. Campbell. *Introduction to remote sensing.* Guilford Press, New York, 1987.

[Campbell 89] W. J. Campbell, R. F. Cromp, and S. E. Hill. Automatic labeling and characterization of objects using artificial neural networks. *Telematics and Informatics*, 6(3–4):259–271, 1989.

[Cromp 91] R. F. Cromp. Automated extraction of metadata from remotely sensed satellite imagery. In *Technical Papers, 1991 ACSM-ASPRS Annual Convention, Volume 3*, pages 91–101. ASCM/ASPRS, 1991.

[Hertz 91] J. Hertz, A. Krogh, and Palmer R. *Introduction to the theory of neural computation.* Addison–Wesley, Redwood City, California, 1991.

[Hecht-Nielsen 90] R. Hecht-Nielsen. *Neurocomputing.* Addison-Wesley, Reading, Massachusetts, 1990.

[Kanal 68] L. Kanal and B. Chandrasekaran. On dimensionality and sample size in statistical pattern recognition. In *Proc. Nat. Electron. Conf.*, pages 2–7, 1968.

[Muhlenbein 90] H. Muhlenbein. Limitations of multi–layer perceptron networks – steps towards genetic neural networks. *Parallel Computing*, 14:249–260, 1990.

[Richards 86] J. A. Richards. *Remote sensing digital image analysis, an introduction.* Springer–Verlag, Berlin, 1986.

[Specht 90] D. Specht. Probabilistic neural networks. *Neural Networks*, 3:109–118, 1990.

[Wan 90] E. A. Wan. Neural network classification: A bayesian interpretation. *IEEE Trans. on Neural Networks*, 1(4):303–305, 1990.

[Weiss 89] S. M. Weiss and I. Kapouleas. An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. In *Eleventh Int. Joint Conference on Artificial Intelligence*, pages 781–787. American Association of Artificial Intelligence, 1989.

[Washburne 91] T.P. Washburne, M. M. Okamura, D. F. Specht, and W. A. Fisher. The Lockheed probabilistic neural network processor. In *International joint conference on neural networks, volume I*, pages 513–518. Institute of Electrical and Electronics Engineers, 1991.

Improved Image Classification with Neural Networks by Fusing Multispectral Signatures with Topological Data[1]

Craig Harston and Chris Schumacher

Computer Application Systems, Inc.
P.O. Box 251
Signal Mountain, TN. 37377
(615) 886-1419
(Fax) 886-7377

ABSTRACT

Automated schemes are needed to classify multi-spectral remotely sensed data. Human intelligence is often required to correctly interpret images from satellites and aircraft. Humans succeed because they use various types of cues about a scene to accurately define the contents of the image. Consequently, it follows that computer techniques that integrate and use different types of information would perform better than single source approaches.

This research illustrated that multispectral signatures and topographical information could be used in concert. Significantly, this dual source tactic classified a remotely sensed image better than the multispectral classification alone. These classifications were accomplished by fusing spectral signatures with topographical information using neural network technology.

A neural network was trained to classify Landsat multi-spectral images of the Black Hills. Bands 4, 5, 6 and 7 were used to generate four classifications based on the spectral signatures. A file of georeferenced ground truth classifications were used as the training criterion. The network was trained to classify urban, agriculture, range and forest with 65.7% correct. Another neural network was programmed and trained to fuse these multispectral signature results with a file of georeferenced altitude data. This topological file contained 10 levels of elevations. When this non-spectral elevation information was fused with the spectral signatures the classifications were improved to 73.7% and 75.7%.

INTRODUCTION

Automated schemes are needed to classify multi-spectral remotely sensed data. For example, the upcoming Earth Observing System (EOS) will generate massive quantities of data that must be managed quickly (Dorfman, 1991; Short, 1991). Access to the resulting data and information should be quick and user friendly. Campbell and Cromp (1990) call for a user friendly system that is based on user domain-specific knowledge and goals. This concept requires that the data system be based on object-oriented storage and retrieval procedures that incorporate information about the image (Dorfman, 1991). Fekete has recommended a sphere quadtree technique for subdividing and relating spherical

[1]This work was supported by the National Aeronautics and Space Administration (NASA) Contract #NAS13-435 with the Stennis Space Center.

data into a data base. This technique relies on the identification of image contents such as coast lines.

If these recommended data base-information systems are to be based on content and knowledge about the data, then real time classification algorithms will be required. Data storage techniques, such as the sphere quadtree (Fekekte) or object oriented information, are based on the contents of the image/data. Data storage will depend on access codes, indexes, or keys specific to the content of data. These codes and indexes would be determined as the data arrives and prior to storage into the data base. An accurate and automated classification technique would be the basis for determining these indexes that will be used for cataloging and filing data. Due to the large quantity of data coming from the EOS, this classification-indexing and storage process should occur in real time or near real time to avoid building a backlog. Not only will it be necessary to transmit and store EOS data efficiently, but also EOS data should be categorized somehow during the transmit or storage process.

While EOS data management will be important, rapid or near real time multispectral remotely sensed data classification is important in its own right. There are potential satellite image applications that depend on rapid access to classification results. Images should be classified without the delay associated with most processing techniques. The results would be transmitted to the user in a timely fashion. This rapid classification and delivery would support the feasibility of many new applications. For example, fishermen could respond quickly to recent current shifts. Short term illegal wild cat mining or deforestation could be identified and arrested. Natural disasters such as oil spills could be monitored as they progress.

Real time classification techniques do not exist; however, neural network technology promises to allow us to automatically classify images in real time. The technique is simple yet can be deployed with parallel neural processing integrated circuits. These processors are relatively cheap and available for multispectral analysis (Harston, Zhant & Stewart, 1991; Kagel, 1991).

The neural network approach has classified various remotely sensed multispectral images (Campbell, Hill & Cromp, 1989; Benediktsson, Swain & Ersoy, 1990; Cromp, 1991; Harston & Schumacher, 1991; Kulkarni, 1990; Eberlein & Yates, 1991; Decatur, 1989). Decatur's work was with the synthetic aperture radar (SAR) HH, HV, and VV components of the return at the L band (1.225 GHz) and the others were with visual and infrared bands. Some of their results can be seen in Table III. While the results compare well with statistical classification techniques, better performance is desirable. It was hypothesized that fusion of spectral signatures with additional information might improve performance.

Human intelligence is often required to correctly interpret images from satellites and aircraft. Humans succeed because they use various types of cues about a scene to accurately define the contents of the image. Consequently, it follows that computer techniques that integrate and use different types of information would perform better than single source approaches. Work to date in our laboratory supports this supposition (Harston, 1991 a, b & c).

This research illustrated that multispectral signatures and

topographical information could be used in concert. Significantly, this dual source tactic classified a remotely sensed image better than the multispectral information alone. These classifications were accomplished by fusing spectral signatures with topographical information using neural network technology.

METHOD

The data came from a Landsat Multispectral Scanner (MSS) image of the Black Hills. Thematic mapper (TM) spectral bands 4, 5, 6 and 7 were represented as intensity values from 0 to 255 in 512 by 512 byte image files. Additionally, files of elevation and ground truth data were available. The ground truth showed that broad contiguous areas were assigned to single classifications. There were 22 potential classifications, which covered urban, farm, range, forest, and water as major groupings. The four classes of data used in this study were urban, farm, range, and forest. These and the other data files were georeferenced.

The type of neural network used was the three layer feedforward networks with one layer as the hidden layer. The delta rule was used to train the output layer and backpropagation was used to train the hidden layer. All work was done on MS-DOS 386/486 VGA microcomputers and the code was written in C.

One neural network was trained to classify Landsat multi-spectral images of the Black Hills. Bands 4, 5, 6, and 7 were used to generate four classifications based on the spectral signatures. These classes included or collapsed several of the classifications found in the ground truth into urban, farm, range, and forest categories. The file of condensed georeferenced ground truth classifications was used as the training criterion. This network was called the spectral signature network.

Another neural network was trained with the four bands of TM data and a topography file of altitude or elevation data. This topological file contained 10 levels of elevations. These images/files were georeferenced to each other, and the ground truth file was used for training. This network was referred to as the fusion network.

Training for both networks consisted of hand picked samples from the larger image. The experiment was conducted twice, resulting in two spectral signature networks and two fusion networks. The second set of networks was tested with additional samples taken from the same image. These samples were taken from intersection points of a grid taken at 50 pixel (horizontal) and 25 pixel (vertical) locations on the upper part of the image. There were 25 test points taken from urban, farm, range, and forest areas that resulted in only one range and one urban testing sample. This kind of grid sampling and random sampling may be roughly representative of the types of data in the image but does not obtain equal numbers of cases for each category.

RESULTS

Both spectral signature networks learned 65.7% of the training sets (60,021 training trials for the second network). The first fusion network learned 73.7% of the training set, and the second fusion network learned 75.8% at 63,035 training trails. Further training resulted in decreased levels of performance.

The second set of networks, both spectral and fusion, were tested with other data taken from the multispectral image. The spectral signature network generalized to these novel data points at 52%, and the fusion network correctly classified 60% of these test cases.

These results (see Table I for results) were carefully reviewed with the image in view. Sixteen percent of the errors appeared to be correctly classified. That is, the ground truth did not appear to be correct from the visual examination of the image. Additionally, 4% of the errors were not clear from the visual image, and the ground truth classification could be debated. The results improved 16% for both the signature and fusion network test results when the scores were corrected for the obvious (not the 4% border line) ground truth errors.

Regardless of the corrections, it is clear that the fusion of altitude information with spectral signatures improved the learning. This improvement was 8% in the first set of networks and 10% in the second set of networks. Even the testing results improved by 8% with the second set of networks.

A detailed analysis of the errors indicated that the greatest number of errors came from misclassified farm data. Keep in mind that there were more test samples from farm areas than from other areas. The performance in each class can be seen in Table II. There was only one range test sample and that one was misclassified. This resulted in a 100% error rate for the range class.

The fusion with elevation data improved the farm scores from 60% to 80%. Unfortunately the forest performance was decreased from 87.5% to 75%. The test sample size was small at 25 cases so interpretation of the results may be limited. The misclassified range sample was one of the debatable or border line cases. These results are also found in Table II.

CONCLUSIONS

The use of altitude data with the spectral signatures improved the performance. This fusion of image and topographic data was simple to do with the neural networks. The elevation data improved the farm land classification but degraded the forest classification to a lesser extent. However, the results were positive overall and suggest that classification performance could be further improved if other types of data were included in the neural classification process.

The initial impression that the learning and test results were low should be interpreted in relationship to the results from similar classifications. For example, as seen in Table III, other types of statistical classification are also low (Benediktsson, Swain & Ersoy, 1990; Duda & Hart, 1973). In general, the neural techniques performed better, except with the multisource technique that used information in addition to the spectral data (Benediktsson, Swain & Ersoy, 1990). This multisource statistical technique included Landsat MSS, elevation, slope, and aspect data. This additional data improved the classification technique to 61%. Clearly, this result argues for the fusion with, or inclusion of, additional cues, regardless of the classification technique used.

Classification of raw spectral data without any clean up is also poor as seen in Table I with 55% (Campbell, Hill & Cromp, 1989) and 52% or 60% in this study. Neural

studies often correct or select the data in some way. Campbell, Hill, & Cromp (1989) used only non-boundary pixels for training. Homogeneous fields were developed for training and testing by Benediktsson, Swain, & Ersoy, 1990. In the present case, the classification was corrected by visual inspection of the test cases. Some of these selection or correction procedures resulted in respectable test scored at 70% (Campbell, Hill & Cromp, 1990) and 76% in this study.

Given the improved classifications by fusing non spectral data with the spectral signature, possibly other supplementary information can be used by the neural network system to improve performance. A shadow file might be used to improve the classification of forests on both sides of the mountain. In another study, pixel patterns based on brightness and texture were classified within each MSS TM band (Harston, 1991d). These classifications were fused with an additional neural network that resulted in improved performance. Possibly, the texture, signature, altitude, and other data can be fused with neural technology to obtain even higher test performance.

The potential for classifying incoming Earth Observing System (EOS) data in real time is genuine. See papers authored by Short, Campbell, Fekete, Dorfman, and Cromp at the GSFC for a description of the importance of this problem. As indicated in our multi-spectral work to date, meaningful results are possible; however, higher levels of performance may be possible. It seems reasonable that more can be done with multi-spectral data when additional non-spectral information is integrated with the spectral results. Further work with seasonal, urban, hazy, and cloudy images is needed. Ultimately, a system that could classify regardless of variations or conditions could categorize incoming data in real time. Such categorizations would be useful for the Intelligent Data Management (IDM) project as a basis for defining, cataloging, and referencing images for a data base.

Acknowledgements: The data was supplied by the Goddard Space Flight Center. Thanks goes to Nick Short and Semir Chettri at GSFC for their assistance.

REFERENCES

Benediktsson, J.A., Swain, P.H. & Ersoy, O.K., (1990). Neural Network approaches versus statistical methods in classification of multisource remote sensing data, IEEE Transactions on Geoscience and Remote Sensing, 28,4, 540-551.

Campbell, W.J. & Cromp, R.F., (1990). Evolution of an intelligent information fusion system, Photogrammetric Engineering and Remote Sensing, 56(6), 867-870.

Campbell, W.J., Hill, S.E. & Cromp, R.F., (1989). Automatic labeling and characterization of objects using artificial neural networks, Telematics and Informatics, 6,3/4, 259-271.

Cromp, R.F., (1991). Automated extraction of metadata from remotely sensed satellite imagery, 1991 ACSM/ASPRS/AUTO-CARTO 10 Proceeding.

Decatur, S.E., (1989). Application of neural networks to terrain classification, IJCNN, I-283-288.

Dorfman, E., (1991). Architecture of a large object-oriented database for remotely sensed data, ACSM/ASPRS/Auto Carto 10 Conference, March, Baltimore.

Duda, R.O., & Hart, P.E., (1973). Pattern Classification and Scene Analysis, John Wiley &

Sons, New York.

Eberlein, S. & Yates, G., (1991). Neural network-based system for autonomous data analysis and control, Progress in Neural Networks, (Ed. Omid Omidvar) Vol. 1, 25-55.

Fekete, G., Rendering and Managing spherical data with sphere quadtrees, available from NASA/GSFC, Greenbelt, MD 20771.

Harston, C.T., (1991a). A Neural Network systems Approach to Image Processing, IEEE Int'l Conf. on Sys., Man, and Cybernetics, Oct. 13-16, 1539-1544.

Harston, C.T., (1991b). Pattern Identification with a Computerized Neural Network, IEEE Southeastcon '91, Williamsburg, April 7-10, 935-939.

Harston, C.T., (1991c). Spacial classification and multi-spectral fusion with neural networks, Proceedings: ANNA'91 Analysis of Neural Network Applications Conference, May 29-31, George Mason University, 76-82.

Harston, C.T. (1991d). The integration of spacial and Multi-spectral data with neural networks, Intelligent Engineering Systems Through Artificial Neural Networks, (Eds. C.H. Dagli, S.R.T. Kumara & Y.C. Shin), ASME Press, New York, 435-440.

Harston, C.T. & Schumacher, C., (1991). Feature extraction and fusion with spectral signatures by neural networks, (in preparation).

Harston, C.T., Zhant, G. & Stewart, D.W., (1991). Neural Network Hardware, part of Phase I SBIR final Report to NASA at the SCC.

Kagel, J.H., (1991). Hardware implementation of a neural network performing multispectral image fusion. SPIE's International Symposium on Optical Engineering and Photonics in Aerospace Sensing, Technical Conference 1469, April 1-5, Orlando.

Kulkarni, A.D., (1991). Neural Networks for Pattern Recognition, Progress in Neural Networks, (Ed. Omid Omidvar), V.1, 197-219.

Short, N. (1991). A Real-time expert system and neural network for the classification of remotely sensed data, 1991 ACSM/ASPRS/Auto-Carto 10 Proceeding, March, Baltimore, MD.

TABLE I

NEURAL NETWORK TRAINING AND TESTING RESULTS

	TRIALS	TRAIN	TEST	CORRECT
EXPERIMENT I.				
SPECTRAL NN		65.7%		
FUSION NN		73.7%		
EXPERIMENT II.				
SPECTRAL NN	60,021	65.7%	52%	68%
FUSION NN	63,035	75.8%	60%	76%

TABLE II

CORRECTED PERFORMANCE FOR EACH CLASS

SPECTRAL SIGNATURE NEURAL NETWORK

CORRECTED GROUND TRUTH	NEURAL NETWORK CLASSIFICATION			
	URBAN	FARM	RANGE	FOREST
URBAN	100%	-	-	-
FARM	26.8%	60%	6.7%	6.7%
RANGE	-	100%	0%	-
FOREST	-	-	12.5%	87.5%

ELEVATION FUSION NEURAL NETWORK

CORRECTED GROUND TRUTH	NEURAL NETWORK CLASSIFICATION			
	URBAN	FARM	RANGE	FOREST
URBAN	100%	-	-	-
FARM	20%	80%	-	-
RANGE	-	100%	0%	-
FOREST	-	12.5%	12.5%	75%

TABLE III

MULTISPECTRAL IMAGE CLASSIFICATION PERFORMANCES

		NEURAL NETWORKS		STATISTICAL CLASSIFICATION *			
		TRAINING	TESTING	ED	ML	MD	MULTI-SOURCE
CAMPBELL, HILL & CROMP, 1989	ANY PIXEL	48%	55%				
WASHINGTON, DC W/ GROUND TRUTH	NON-BOUNDRY PIXELS	66%	70%				
BENEDIKTSSON, SWAIN & ERSOY, 1990							
COLORADO MOUNTAINS	HOMOGENEOUS FIELDS	95%	52.5%	47%	49%	50%	61%
CROMP, 1991			60%				
BLACK HILLS LANDSAT MSS							
HARSTON, 1991	MLC W/ NN	100%	61%				
MURFREESBORO LANDSAT MSS MULTISPECTRAL FUSION SYSTEM	1 SAMPLE /CLASS SET	100%					
	BRIGHTNESS	85%	43%				
	BRIGHTNESS & TEXTURE	75%					
HARSTON & SCHUMACHER, 1991	SPECTRAL SIGNATURE W/ ROAD DETECTOR NN SYSTEM	95%	63%				
TIMS IMAGES							
HARSTON & SCHUMACHER, 1991		W/ GROUND TRUTH		CORRECTED FOR VISUAL INTERPRETATION			
BLACK HILLS LANDSAT MSS	SPECTRAL SIGNATURE	65.7%	52%	68%			
	SPECTRAL SIGNATURE W/ALTITUDE	75.8%	60%	76%			

* ED: MINIMUM EUCLIDEAN DISTANCE
ML: MAXIMUM LIKELIHOOD METHOD
MD: MINIMUM MAHALANOBIS DISTANCE
MULTISOURCE: STATISTICAL MULTISOURCE ANALYSIS

Improved Interpretation of Satellite Altimeter Data Using Genetic Algorithms

Kenneth Messa
Department of Mathematical Sciences
Loyola University
New Orleans, LA
(504)865-3340
fax: (504)865-3347

Matthew Lybanon
Naval Oceanographic and Atmospheric Research Laboratory
Stennis Space Center, MS
(601)688-5263

ABSTRACT

Genetic algorithms (GA) are optimization techniques that are based on the mechanics of evolution and natural selection. They take advantage of the power of cumulative selection, in which successive incremental improvements in a solution structure become the basis for continued development. A GA is an iterative procedure that maintains a "population" of "organisms" (candidate solutions). Through successive "generations" (iterations) the population as a whole improves in a simulation of Darwinism's "survival of the fittest". GAs have been shown to be successful where noise significantly reduces the ability of other search techniques to work effectively.

Satellite altimetry provides useful information about oceanographic phenomena. It provides rapid global coverage of the oceans and is not as severely hampered by cloud cover as infrared imagery. Despite these and other benefits, several factors lead to significant difficulty in interpretation.

The GA approach to the improved interpretation of satellite data involves the representation of the ocean surface model as a string of parameters or coefficients from the model. The GA searches in parallel a population of such representations (organisms) to obtain the individual that is best suited to "survive", that is, the fittest as measured with respect to some "fitness" function. The fittest organism is the one that best represents the ocean surface model with respect to the altimeter data.

1. INTRODUCTION

Much useful information about oceanographic phenomena can be obtained from an altimeter borne on a satellite. In addition to providing rapid global coverage of the oceans, satellite altimetry bypasses other (in-situ) measurement problems. It is not as severely hampered by cloud cover as infrared imagery, and it also measures oceanographic phenomena that have no surface thermal expression. Despite the benefits of altimetry, several factors lead to significant difficulty in interpretation. Among these are atmospheric noise (from water vapor, ionospheric electrons, solar activity, and so forth), scale errors (the magnitudes of many of the errors are greater than the phenomena measured), some measurements are time dependent while other related ones are time independent (the presence of the mean dynamic topography in the reference surface or "geoid", for example) and in the calculations of the geoid itself.

In this paper we first present some background on the use of satellite altimetry data to measure the sea surface. The interpretation of these measurements is complicated by the difficulties referred to above. In order to improve our

interpretation of the altimeter data, we turned to a technique based on an optimization procedure believed to operate effectively in nature. Known as "genetic algorithms" (GA), these techniques have been shown to be successful in many environments. Because they search in parallel a large portion of the solution space, they are able to distinguish local optima from global ones. GAs can successfully search where noise significantly reduces the ability of other search techniques to work effectively. We demonstrate the effectiveness of GAs to fit a model of the sea-surface height to data obtained from satellite altimetry.

2. BACKGROUND

A satellite-borne radar altimeter measures the distance from its antenna's electrical center to the instantaneous sea surface, averaged over the footprint. Sea level is the difference between the altimeter-measured distance and the satellite's height; the latter is determined independently by tracking and orbit determination. Then, the difference between sea level and the geoid, the sea surface height (SSH) residual, provides information on ocean dynamics.

The geoid is a graviational equipotential surface. The marine geoid is the shape that would be taken by a resting ocean. Since the geoid does not change, and since the oceanographic component of sea-surface variations is generally relatively small and does change with time, the long-term temporal mean of sea level is a good approximation to the marine geoid.

The situation is different when one tries to use the altimeter to measure ocean circulation. Then the "signal" is the small SSH residual that remains after the geoid is subtracted from sea level, and the "noise" is any error in knowledge of the geoid. For practical reasons, a large part of the information that goes into "geoids" comes from altimeter measurements themselves. When there are permanent oceanographic features such as the Gulf Stream, there is a significant time-independent "dynamic height" component. Being independent, this component cannot be distinguished from the true geoid without additional information. Subtraction of a geoid containing this term from the altimeter-determined sea level may introduce a serious error (Lybanon et al., 1990).

One model for the sea surface height is realized as the difference between the expected dynamic height component and the reference surface error. This model can be tested using GEOSAT altimeter measurements of the Gulf Stream region, which has a strong mesoscale signal (Calman, 1987). The following model has been proposed by Lybanon et al. (1990):

$$SSH = A\tanh[B(X-D-E)] - F\tanh[C(X-E)] - G \qquad (1)$$

where X is the along-track coordinate of the satellite. The first term represents the instantaneous Gulf Stream, the second represents the mean Gulf Stream, and the third term is an overall bias due to orbit error or possibly other errors. The coefficients in each hyperbolic tangent represent the amplitude, the steepness of the sloping part, and the position of the curve, respectively. By fitting this model to the altimeter data, one can add the modeled mean Gulf Stream profile to the instantaneous sea surface height to allow a better description of the Gulf Stream. The key to this proposed technique is finding the coefficients from Equation (1) that best fit the altimeter data. Lybanon et al. (1988) have attempted to use standard mathematical curve-fitting schemes, but have achieved only mixed success. We propose using GAs to aid this process.

3. GENETIC ALGORITHMS

Genetic algorithms are optimization techniques that are based on the mechanics of evolution and natural selection. In contrast to other methods that rely on a point-to-point search of the domain space, GAs use a large sample of points from the domain. Each point, called an "organism", is a candidate solution of the problem in question. The large sample of candidate solutions (called a "population") is modified through successive iterations. Each modification is based on ideas taken from Darwinian natural selection. Although randomness is a part of the process, each modification is guided by the candidate solutions that are most successful. These "fittest" organisms

contribute the most to succeeding iterations in a simulation of "survival of the fittest". Each successive population is called a "generation". Thus we have an intitial generation, G(0), and for each generation G(t), the GA forms a new one, G(t+1). An algorithm to implement GAs is given by:

```
generate initial population, G(0);
evaluate G(0);
t := 0;
repeat
        generate G(t+1) using G(t);
        evaluate G(t+1);
        t := t+1;
until solution is found.
```

Like all generate and test methods, the GA requires the two main steps of generation and evaluation. In order to evaluate a generation, a fitness function is needed. In nature, a species responds in some way to environmental pressure. The GA analog to this pressure is the fitness function. It is built from domain specific information and returns the relative merit or fitness of the organism (Goldberg, 1989).

3.1 Representation

Our problem entails finding coefficients A, B, ..., G which yield the best fit of the altimeter data when used in Equation (1). The measurement of the goodness of fit with respect to the data $\mathbf{D}$ is the "fitness function". Since we are searching for real number values for the coefficients A, B, ..., G, the organisms for the GA used here are vectors $r = \langle r_A, r_B, \ldots, r_G \rangle$ of real numbers. The fitness of such an organism is the degree to which the model equation SSH(r) successfully fits the data.

This view of the representation is useful at the higher level of the curve-fitting problem. However, the genetic algorithm works at a lower level—the level of bits. In order to successfully use the GA, we need to consider a representation of the real numbers r_i at the bit level. Given upper and lower bounds for each r_i, u_i and l_i respectively, we can look at r_i as an unsigned binary integer with m bits and calculate its value with respect to l_i and u_i.

Given a binary integer b where b is in $[0, 2^k - 1]$, we can derive its corresponding real value using the formula

$$r = b/2^m * (u - l) + l \qquad (2)$$

where u and l are the upper and lower bounds respectively. Combining these two levels, we construct an organism as $\theta = \langle b_{A1}\ b_{A2} \ldots b_{Am}, b_{B1}\ b_{B2} \ldots b_{Bm}, \ldots, b_{G1}\ b_{G2} \ldots b_{Gm} \rangle$ where each binary integer $b_{i1}\ b_{i2} \ldots b_{im}$ corresponds to a real number r_i which lies in the interval $[l_i, u_i]$ for i = A, B, ... G. The correspondence is given in (2).

Computing the value of the fitness function of an organism θ requires two steps: first, converting each binary integer $b_{i1}\ b_{i2} \ldots b_{im}$ into its corresponding real value r_i; then, second, evaluating the curve $SSH(r_A, r_B, \ldots, r_G)$ at the data points of $\mathbf{D}$.

3.2 Evaluation

Since the fitness function is a measurement of how well the organism fits the data, it focuses the GA toward the solution. The fitness function used here is modeled after least squares/regression. An organism θ is converted into a vector of real numbers, $r = \langle r_A, r_B, \ldots, r_G \rangle$ using equation (2). The fitness is then computed as the sum of the squares of the differences between the $SSH(r;x_i)$ and y_i (that is, the residuals). Thus,

$$\text{fitness}(\theta) = \Sigma\ (SSH(r;x_i) - y_i)^2 \qquad (3)$$

where the summation is a taken over all data points (x_i, y_i) of $\mathbf{D}$. With this fitness function, a value of 0 is considered a perfect fit and an organism is highly fit if its fitness value is low.

3.3 Convergence

The GA is designed to improve the relative merit of the population over time. While the average fitness of one generation may be lower than the preceeding one, or while the best solution from one generation may not be as good as the best from a previous one, in general, fitnesses improve as generations unfold. Figure 1 in the appendix is an example of this point.

In earlier generations, there is a great deal of variability among the organisms in a single generation. There is a wide range of fitness values in these earlier generations. As happens occasionally, a few organisms are generated whose fitnesses are exceedingly poor. This reduces the average fitness of the overall generation. The number of poor solutions generated is in proportion to the fitness of the generation as a whole. Thus, in the early generations, a larger number of poor solutions are formed. However, as the generations improve and larger numbers of the organisms have good fitnesses, the occasional poor performer does not affect the population as much. The effect of these less fit organisms in later generations is minimal.

As the generations improve, the average fitness stabilizes. As a result, most of the organisms are nearly identical. This stabilization is called "convergence", and the GA is said to converge to the organism that appears most often. Of course, at convergence, nearly all organisms are identical. This commonality is the solution to the problem.

3.4 Operators

GAs are based on many of the same principles as those found in natural selection. They employ several operators and principles that are generally derived form those that occur in nature. There are three principal operators at work in GAs: selection, crossover and mutation. The first of these, selection, is the analog of the principle in natural selection that organisms that are most fit are most favorably disposed for participation in mating, thereby passing their genetic information to their offspring. The selection operator chooses individuals from the population so that those with high fitnesses have greater probability of being selected. This focus toward the highly fit individuals is what drives genetic algorithms.

The method of selection that was used here is stochastic sampling without replacement, called "expected value" by Goldberg (1989). In addition, we have used de Jong's (de Jong 1975) "elitist" strategy, whereby the single best organism form one generation is placed unchanged into the next generation. This strategy gives a little more weight to the best organism than might be achieved from selection alone and prevents the possibility that the best organism might be lost early through crossover or mutation.

The GA analogy to mating is called crossover. The crossover operator provides a mixing of the genetic material from the parents, and globally, it mixes the genetic information of the whole population. It is the mixing of the "genes", the stirring of the pot of genetic material, that gives the GA robustness. The two organisms chosen by selection are combined to form a new individual with similarities to both parents. If the mixing is done carefully, a large amount of genetic material can be tested. Although selection focuses the genetic algorithm, it is crossover that adds variety.

We employed a "two-point" crossover. Two-point crossover proceeds as follows: Once the organisms (the "parents") have been selected for mating, two bit positions are chosen at random. The middle segments between these bit positions of the two organisms are interchanged to form two new organisms. These new organisms (the "offspring") are added to the next generation. The process of selection and crossover is repeated until the new generation has the same number of members as the previous generation.

While selection and crossover are the chief operators used in GAs, there are numerous other minor operators proposed to strengthen GAs under certain circumstances. It has been shown that for certain applications, these minor operators can add to the GA's efficiency or prevent it from converging to a local optimum rather than global one. For example, it sometimes happens that the GA converges to a

solution prematurely. This is due to the fact that crossover only mixes the genetic material that is present in the initial population; it doesn't introduce any new material. In nature, new genes are introduced into a species through mutation. Analogously in GAs, a mutation operator is used to modify an organism occasionally in order to add new genetic material into the population and to prevent premature convergence.

We used a mutation method that adds a real value ϵ to (or subtracts it from) the organism's value at one of the coefficents. We kept the probability of a mutation low. Thus, if by chance a particuplar organism was to be mutated at one of its coefficients, r, then a small ϵ_i was added to (or subtracted from) r. The value of ϵ is a power of 2 ranging from 1 to 2^m. Thus, if we let $\epsilon_i = < 0 \ldots 0\ 1\ 0 \ldots 0 >$ with 1 in the i-th position and 0's elsewhere, then this mutation method effectively adds ϵ_i to (or subtracts ϵ_i from) the coefficient of the organism to be mutated.

4. RESULTS

The GA technique outlined above is dependent on the choice of boundary points l_i and r_i for each i = A, B, ..., G. Knowledge of the problem domain may by useful to ascertain these boundaries. If the knowledge is inadequate, a degree of experimentation may be required. Our knowledge of the problem, gained in part by previous work (Lybanon et al., 1988), gave us some information about the coefficients. We knew that the amplitudes of the curves (coefficients A and F) were positive and less than 1. Likewise we knew the slope coefficients (B and C) were also positive and less than 1. Hence, for all these values, we used domain intervals [0, 1]. The error of G was small, but its sign was unknown to us. We used [-1, 1] as its domain interval. The difficult values to determine were the position coefficients D and E.

We began with intervals of length 100 for each of D and E. We were prepared to test several intervals of this length, [-50, 50], [-75, 25], [-25, 75], and so forth. If necessary, we might have had to increase the length of the intervals to 200 or more. In any case, we were prepared to do the experimentation needed for the GA. The results given below show that we needed only the second interval mentioned.

After several runs using the domain interval [-50, 50] for coefficents D and E, it became clear that this domain did not include the value for E, and perhaps not D either. One run placed the optimal E exactly at -50, indicating either a coincidence that we chose an endpoint of the domain interval very near the optimal value or the possibility that E lay below -50. The other runs had somewhat low errors, yet the values of the coefficients were not near each other. This could also mean that we stopped the GA too soon, before it had a chance to converge. See Table 1. On the strength of the former observation, we abandoned the [-50, 50] interval for E in favor of [-75, 25]. If indeed -50 was the optimal value for E, this new interval would bear this out. If the error was due to the optimal E being smaller than -50, then this or another interval would be better. If the results for the new interval were likewise inconsistent, we would allow the GA to run for several generations longer and compare results. Since D did not seem to suffer the same error as E, we were not quick to adjust D's domain interval. Table 2 exhibits the results obtained with these new intervals. The value of E derived here supports our earlier decision to reduce the lower bound for E's interval.

Table 1: Coefficients obtained with intervals [-50, 50] for D and E.

coefficient	run # 1	run # 2	run # 3
A	.273373	.191273	.240241
B	.435305	.499703	.504699
C	.058454	.008573	.008887
D	15.7059	-46.5097	-40.3814
E	-50.0000	13.6979	7.37317
F	.260792	.640602	.851496
G	-.028586	-.352537	-.414537
error	.612744	.903165	.825254

Table 2. Coefficients obtained with intervals [-75, 25] for E and [-50, 50] for D.

coefficient	run # 1	run # 2	run # 3
A	.186314	.186297	.186302
B	.757704	.757711	.757642
C	.174833	.174889	.174872
D	24.7986	24.8003	24.7997
E	-58.7977	-58.7995	-58.7989
F	.162858	.162844	.162848
G	-.025868	-.025869	-.025868
error	.558747	.558747	.558747

Table 3. Coefficients obtained after several stages of reducing the intervals

coefficient	run # 1	run # 2	run # 3
A	.186290	.186290	.186290
B	.757685	.757680	.757682
C	.174907	.174907	.174907
D	24.8009	24.8009	24.8009
E	-58.8000	-58.8000	-58.8000
F	.162838	.162838	.162838
G	-.025870	-.025870	-.025870
error	.558747	.558747	.558747

To contrast these results, we tried the interval [-25, 75]. We obtained inconsistencies in all runs, as might be expected, as the correct value of E was far removed from this interval.

In this problem, we had some knowledge of the coefficients. However, one can obtain very accurate results with very little knowledge, if one is allowed to experiment. Beginning with a domain interval of [-100, 100] for each coefficent, after several runs we were able to revise our results for A, F, and G as lying in the interval [-5, 5]. The experiments gave us no information about B, C, D, or E, however. We ran the GA several more times on the revised intervals. At this point we were able to further narrow the intervals for some of the coefficients. By repeating this process through just four stages of interval reduction, we were able to obtain the results in Table 3.

When small enough intervals are used (gained through either experimentation or knowledge of the model), one can get a very accurate fit of the data. With an accurate fit, the dynamic height component can be removed yielding a more accurate interpretation of the data. See Figure 2 in the appendix. It shows the original altimeter data of the Gulf Stream and the adjusted values after the dynamic component has been removed.

5. SUMMARY

We have demonstrated that genetic algorithms can be used successfully to improve the interpretation of altimeter data in a model for the sea surface height. There are several strengths to this approach. First, it does not require complex calculation nor is it difficult to set up. Second, it is accurate in its present form. With 32 bit representation of integers, we easily obtained 4, 5, or 6 significant digits. More accuracy can be achieved with minor revisions. Finally, the results were consistent, although the initial genetic algorithm parameters needed to be established at the beginning.

The method suffers from some weaknesses, however. First, the initial set of parameters is not univerally known. There must be some experimentation on these parameters initially and more experimentation on these if there is a significant change in the structure of the model. Second, the method requires some knowledge of the model coefficients. This knowledge can be gained through inspection of the data or of the function itself, or it can be gained through experimentation. In either case, the genetic algorithm method is a viable technique for improving the interpretation of the altimetry data used in this model.

6. ACKNOWLEDGEMENTS

This research was sponsored by the U. S. Navy Office of Naval Technology, CDR Lee Bounds, Program Manager, and by the Space and Naval Warfare Systems Command, LCDR William Cook, Program Manager.

Kenneth Messa's work was partially supported by the Naval Oceanographic and Atmospheric Research Laboratory through the U. S. Navy/ASEE Summer Faculty Research Program.

This is NOARL Contribution Number PR 91:121:321. It is approved for public release; distribution is unlimited

7. REFERENCES

Baker, J. E. (1987). Reducing bias and inefficiency in the selection algorithm. *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, 14-21.

Calman, J. (1987). Introduction to sea-surface topography from satellite altimetry. *Johns Hopkins APL Technical Digest 8*(2), 206-211.

Daniel, C. & Wood, F. S. (1980). *Fitting Equations to Data*, 2nd Ed. New York: John Wiley & Sons.

De Jong, K. A. (1975). An analysis of the behavior of a class of genetic adaptive systems. (Doctoral dissertation, University of Michigan). *Dissertation Abstracts International 36*(10), 5104B.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley.

Grefenstette, J. (1986). Optimization of control paramenters for genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-16*(1), Jan/Feb 1986.

Guest, P. G. (1961). *Numerical Methods of Curve Fitting*. Bristol, Great Britain: Cambridge University Press.

Holland, J. (1975). *Adaption in Natural and Artificial Systems*, Ann Arbor, Michigan: University of Michigan Press.

Lybanon, M. & Messa, K. (1991). Genetic Algorithm Simulation to Improve Altimetric Sea-Surface Height Residuals, to appear in *Proceedings of SimTec '91*.

Lybanon, M., Crout, R., Johnson, C. & Pistek, P. (1990). Operational altimeter-derived oceanographic information: *The NORDA GEOSAT ocean applications program, Journal of Atmospheric and Oceanic Technology, 7*(3), 357-376.

Lybanon, M., Johnson, D. R. & Romalewski, R. S. (1988). Separation of the mean Gulf Stream topography from an altimeter-derived reference surface, *EOS Transactions, American Geophysical Union, (69)*44, 1281.

Lybanon, M. & Crout, R. L. (1987). The NORDA GEOSAT ocean applications program, *Johns Hopkins APL Technical Digest, (8)*2.

Suh, J. Y. & Van Gucht, D. (1987). Incorporating information into genetic search. *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, 100-107.

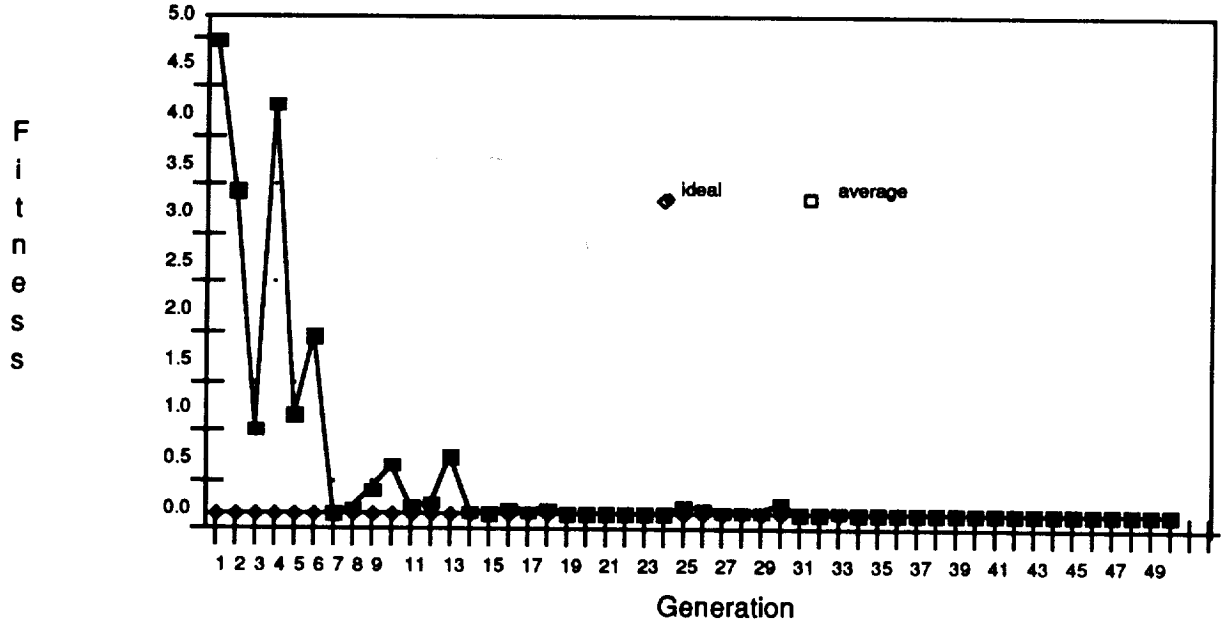


Figure 1 Average fitness of a population demonstrating convergence in a Genetic Algorithm

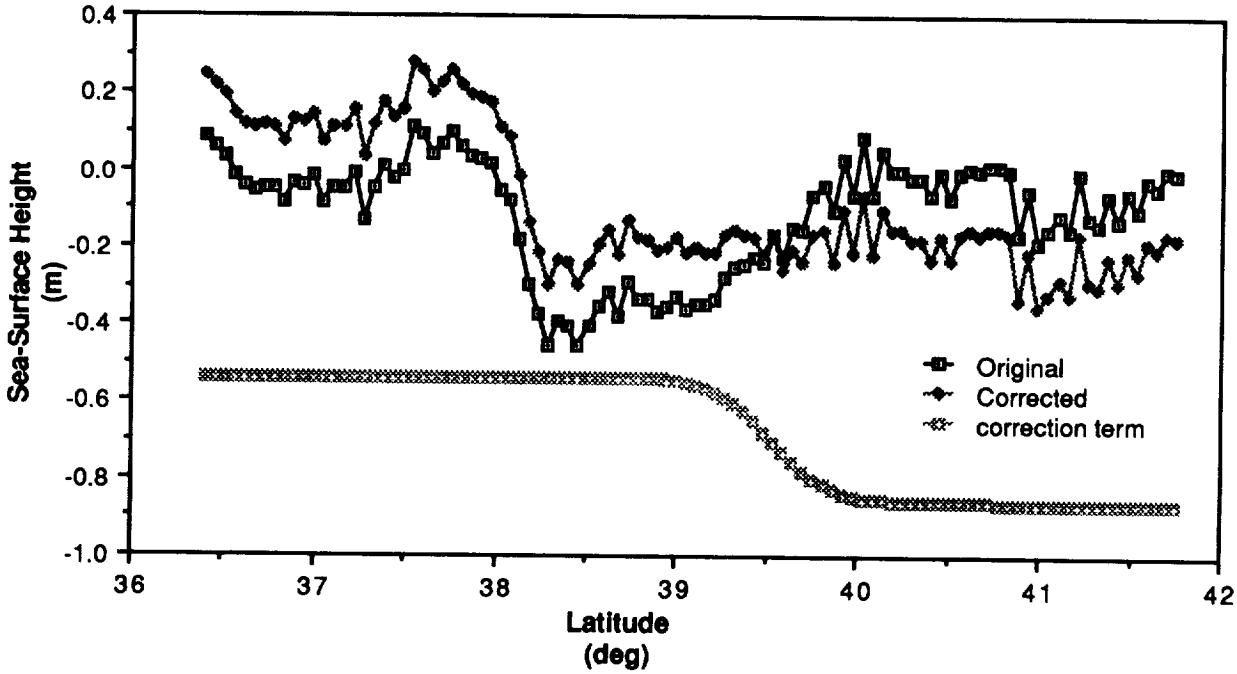


Figure 2 Original altimeter data and Correction. Correction term has been displaced for easier viewing.

Knowledge Engineering

The Use of Artificial Intelligence Techniques to Improve the Multiple Payload Integration Process

Dannie E. Cutts [1]
Space Operations Department
Teledyne Brown Engineering
Huntsville, Alabama

Brian K. Widgren
Manager, CDMS Integration
Payload Integration Engineering Department
Teledyne Brown Engineering
Huntsville, Alabama

Abstract

A maximum return of science and products with a minimum expenditure of time and resources is a major goal of mission payload integration. A critical component then, in successful mission payload integration is the acquisition and analysis of experiment requirements from the Principal Investigator (PI) and Payload Element Developer (PED) teams. This paper describes one effort to use Artificial Intelligence (AI) techniques to improve the acquisition and analysis of experiment requirements within the Payload Integration Process.

[1] The authors may be contacted at:
Teledyne Brown Engineering, MS-172
300 Sparkman Drive NW
Huntsville, Alabama 35807-7007
(205)726-5929

OVERVIEW

As the payload integration contractor to Marshall Space Flight Center for Spacelab payloads, Teledyne Brown Engineering (TBE) has been heavily involved in the acquisition, analysis, and integration of payload requirements for a number of years. NASA/MSFC and TBE are currently involved in efforts to streamline and improve the mission integration process. Part of this improvement effort involves the use of Expert Systems in several areas. A number of benefits are anticipated from the use of these systems, including:

- A better understanding by the PI/PED teams of STS and Spacelab capabilities,
- On-line help and documentation capabilities,
- On-line data validation rules to check data for accuracy and reasonableness,
- A more consistent approach to experiment requirements gathering and analysis,
- Increased quality in the requirements definition data provided to the integration team, resulting in fewer iterations between the PI/PED and integration teams,
- Increased quality in the analyses performed on those requirements,
- Reduction in the need for members of the integration team to travel to PI/PED sites,
- Retention and documentation of corporate expertise in payload integration, and
- Training tools for Payload Integration Engineers.

Throughout the Spacelab integration process, averaging about 42 months from Authority to Proceed (ATP) to the actual mission, a series of readiness reviews are held to ensure that requirements definition and integration are progressing on schedule. The current approach to gathering experiment requirements is human intensive requiring numerous iterations between the PI/PED/Integration teams [FIG 1].

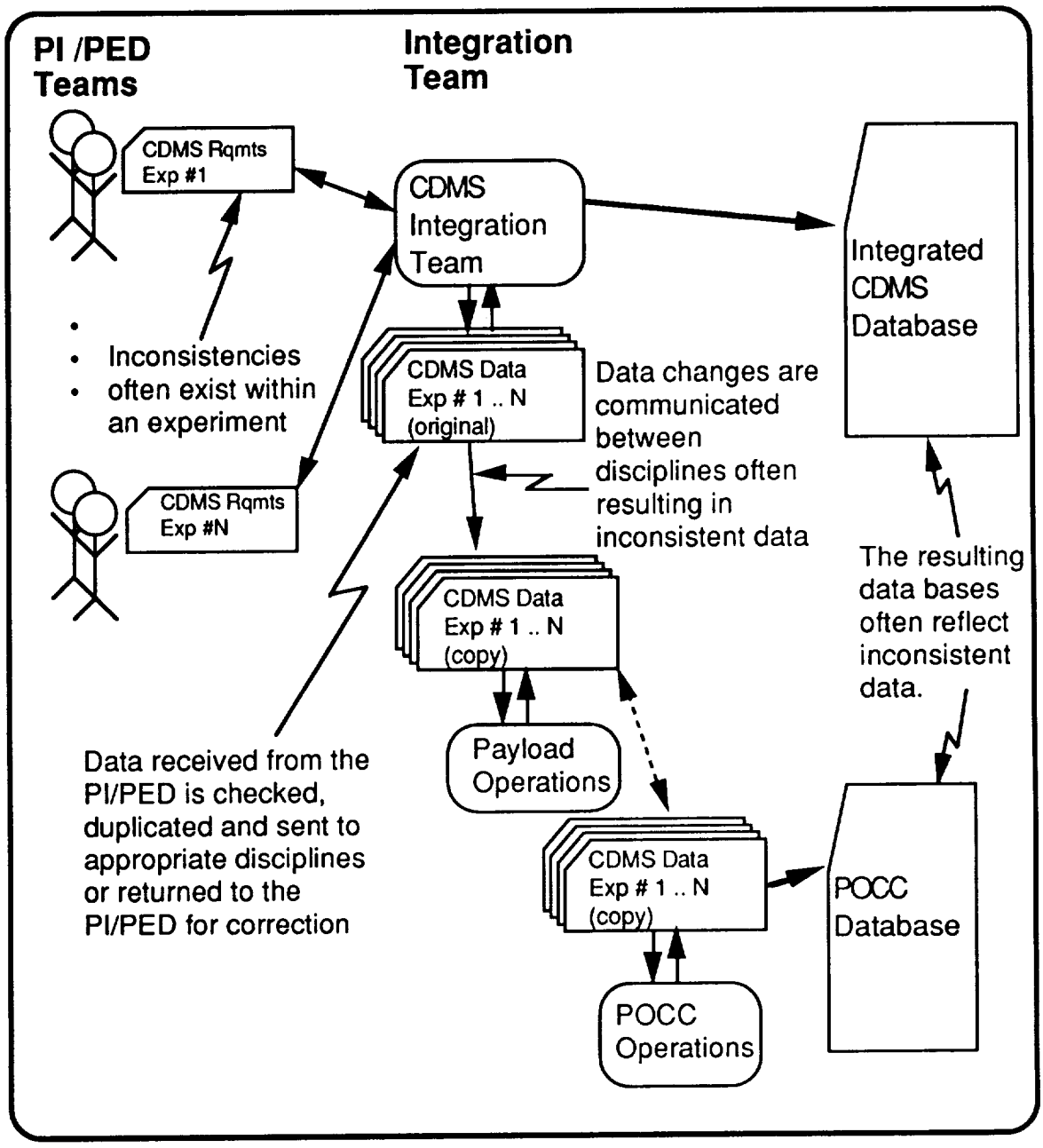
Much of this iteration takes place as a result of changing requirements. A volumous "data pack" of information describing STS and Spacelab capabilities and constraints is sent to each PI/PED. Instructions outlining responsibilities and deadlines are included in the pack along with directions for completing the requirements definition forms. Digesting and interpreting this data pack can present a formidable task for the PI/PED. As a result, requirements are often submitted late, poorly defined, or incomplete. All these situations can significantly affect the readiness reviews and thus the entire integration process.

This paper describes one project within TBE to use Expert Systems to automate portions of the acquisition and analysis effort for the Commanding and Data Management system (CDMS) requirements. The acquisition and analysis of CDMS requirements begin very early in the mission design process and impact nearly every discipline. Historically, changes to CDMS requirements take place throughout the entire life of the mission design cycle. These changes come about for various reasons including users not understanding Spacelab or STS capabilities, changing hardware/software configurations, design reviews, operational considerations as well as problems resulting from the complex interactions between an individual experiment and the integrated payload. The high cost of incorporating changes late in the integration process made the CDMS expert system a "high payback" candidate for development.

CDMS DESCRIPTION

CDMS data consists of a set of PI/PED generated documents defining requirements for on-board data processing and display, telemetry monitoring and experiment commanding by the Spacelab experiment computer as well as downlink requirements and POCC (Payload Operations Control Center) processing requirements.

The POCC supports extensive data processing capabilities including ground monitoring of telemetry data, acquisition and storage of science data, as well as ground commanding of the on-board experiments by the PI.



Current approach to acquiring & analyzing CDMS requirements

[FIG 1]

These requirements are defined in a set of eight tables generated by the PI/PED team and expanded by the integration team and other contractors.

CDMS REQUIREMENTS ACQUISITION

Early in the mission design process, PI/PED teams are required to supply preliminary CDMS requirements. Since the initial submission of these requirements takes place very early in the design phase, some

requirements may not be firm because of unresolved hardware or software issues. The fact that these requirements remain unresolved may or may not be identified in the submitted documents. The integration team may have no way of knowing which requirements are to be supplied later. Often these "missing" requirements are not identified until late in the integration process resulting in reworks or delays. With the automated acquisition tool, PI/PEDs can flag data as "TBS" so that the integration team can follow up on later definition.

Under the current approach for gathering CDMS requirements, the PI/PED supplies requirements in paper format to the CDMS engineers. These engineers then do a manual cross checking of the paper inputs to ensure accuracy and agreement across tables before inputting it into an electronic format. Some errors in the data are missed during the "paper analysis" while others occur as a result of operator entry errors. Errors identified in the PI/PED supplied requirements are returned to the PI/PED for revision. Submission of better initial data from the PI/PED will help to shorten the number of these iterations that take place.

These CDMS data are currently stored as text files on either VAX or Macintosh computers. Various analyses (both manual and automated) are performed on these text files and modifications are made directly to them. After analyses are performed on individual files representing experiments, an integrated CDMS "database" file is constructed by concatenating the individual experiment files. Analyses are then performed on the complete file representing the integrated payload.

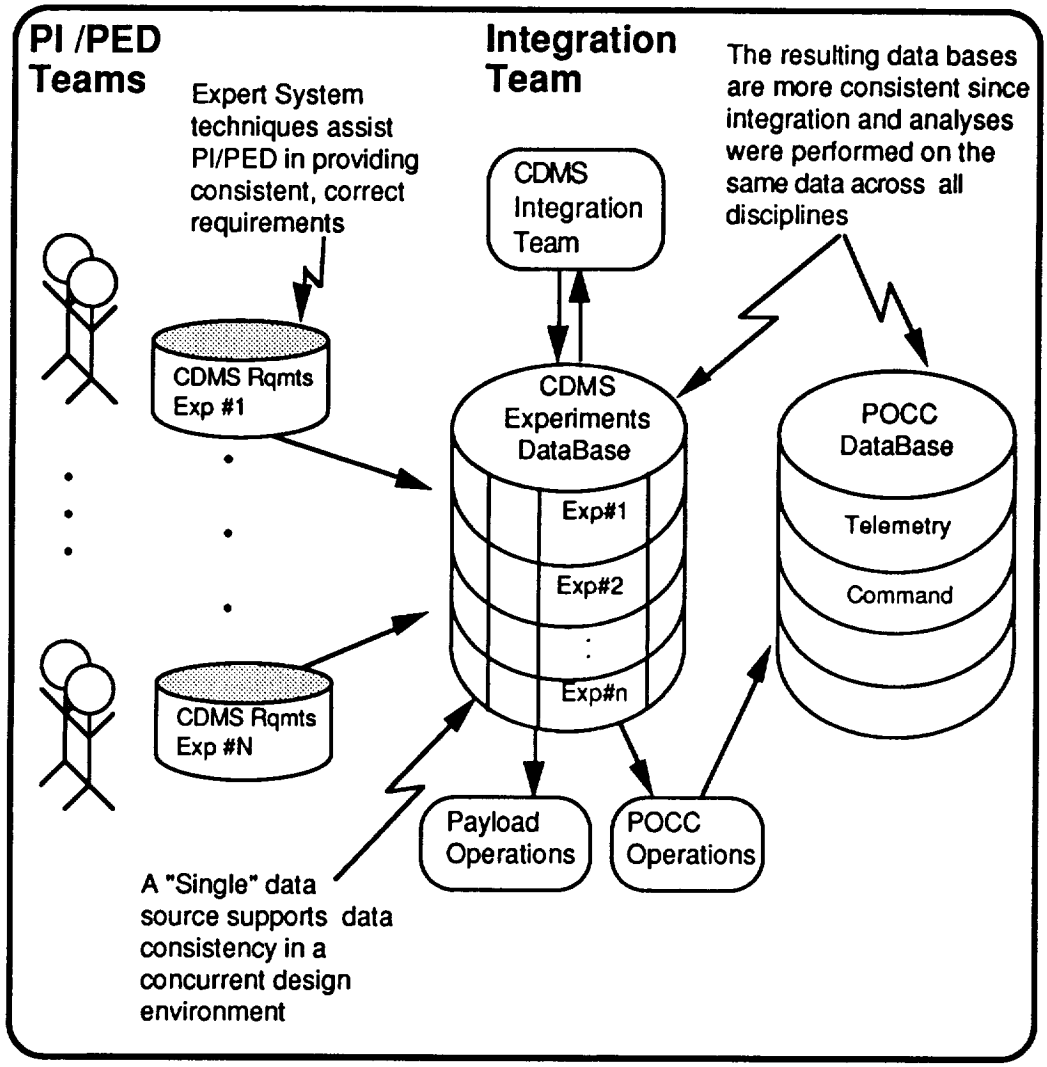
In an effort to improve the above process, we plan to have the PI/PED teams submit CDMS requirements electronically using an acquisition tool [FIG 2] being developed by TBE .

This tool utilizes a number of expert system techniques including limited "object" support, constraint checking, and a simple form of retraction in which data is valid only when supported by other data (Doyle 1981). When the supporting data changes, the "supported" assertion is changed. These "dependency" relations are implemented in a hypertext environment in which data objects contain lists of other data objects on which they are dependent and which are dependent on them. This "dependency" matrix is used to guide the dialogue with the user, provide explanations and support data retraction.

Some problems in the current data requirements definition occur as a result of the user not understanding STS or Spacelab capabilities. To assist in this area, the automated acquisition tool offers explanations of the questions being asked which can include detailed descriptions of hardware or carrier issues. In addition, we have implemented a "why" facility which can provide context sensitive information on why the particular datum is important. This capability is closely tied to the "dependencies" relations mentioned earlier. This acquisition system will assist the PI/PED teams in supplying complete and correct requirements definitions to the integration team.

CDMS REQUIREMENTS ANALYSIS

The second component of this process improvement effort deals with the analysis of the acquired CDMS data. Once the PI/PED generated requirements documents reach the integration team, they must be checked independently for problems and then combined with other experiment requirements files to perform integrated analyses. Error checking in the automated requirements acquisition system described above ensures that the data arrives relatively error free. Since the documents are submitted electronically by the PI/PED teams, errors introduced during the data input phase are eliminated. Engineers are also freed from the task of inputting the PI/PED data. The current text file storage of CDMS requirements data is being replaced with a relational database. Having these data stored in a relational database allows the integration team to more easily manipulate the CDMS data.

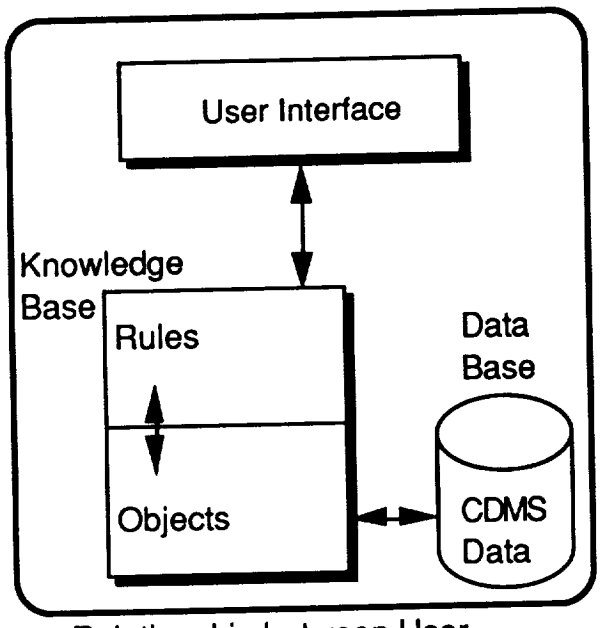


Automated approach to acquiring and analyzing CDMS requirements

[FIG 2]

As these analyses progress, the integration team adds a significant amount of mission dependent data. The data supplied by the integration team is also automatically checked as it is entered by the engineers.

As mentioned earlier, CDMS data impacts numerous other disciplines within the integration team. In the past, a significant amount of time was spent by the CDMS group in inputting and checking PI/PED data before that data could be made available to other disciplines. Under the automated acquisition and analysis approach, the data arrives in electronic format with significant data checking already performed. As a result, portions of the data can be made available to other disciplines much faster. This analysis tool is used to transfer the PI/PED requirements from the electronic file format submitted by the PI/PED to a relational database. The user interface of the analysis tool also allows the integration team to manipulate the data and generate various report formats. Rules are used to access the data stored in the relational database to perform validity checks within and across experiments [FIG 3]. Both mission dependent and mission independent checks are made.



Relationship between User, Domain Knowledge & Data

[FIG 3]

REQUIREMENTS DATABASE

In support of this automated acquisition and analysis effort, TBE is establishing an integrated "requirements database" using Oracle. This integrated requirements database will provide a single source of experiment data from which all members of the integration team will work. This will eliminate the problem of different teams working with different versions of the same data. Impacted users can be automatically notified when data has been changed. We also are planning to use the integrated database to generate portions of deliverables (including documentation) for use within TBE and NASA as well as other subcontractors. It is anticipated that some of the applications accessing the integrated requirements database will utilize expert systems while others will be conventional systems.

METHODOLOGY

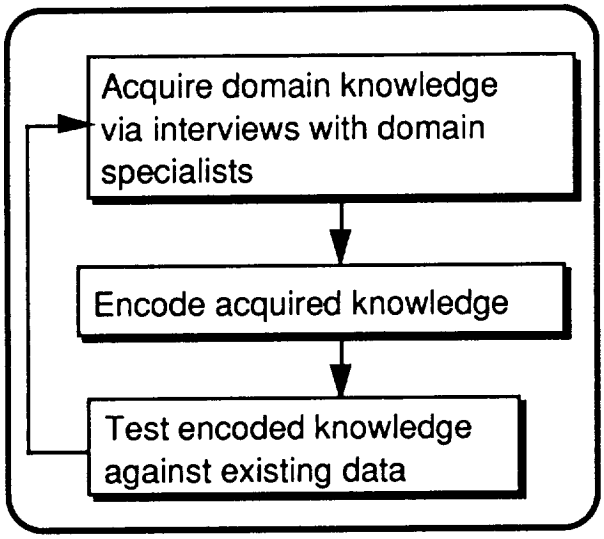
In the design and development of both the acquisition and analysis applications a number of issues arose. They are listed and briefly discussed below:

1. Knowledge Acquisition

Most experts agree that knowledge acquisition is one of the most (if not THE most) critical components in the development of knowledge based systems - and often the most labor intensive (Gaines 1989). Although tools exist for the automated acquisition of domain knowledge, we chose to use manual interview techniques for this effort since the knowledge acquisition task in this case was relatively straightforward. A great deal of knowledge about data relationships is already documented in a set of detailed instructions to the PI/PED supplying CDMS requirements (MDC 1991). This document covers only the on-board CDMS data requirements (four of the eight tables populated by the PI/PED), but did provide an excellent starting point for acquiring domain knowledge. There were several reasons why this document provided only a starting point, and not the entire knowledge base. First, the document only defines data relationships and constraints. This type of knowledge, while important, does not support any type of intelligent dialogue between the system and the PI/PED user. Also missing were heuristics and explanation knowledge of how a human expert would query the PI/PED to acquire CDMS data. This control knowledge will be discussed later.

Since the PI/PED customarily provides paper copies of CDMS requirements and no human is usually involved in the initial acquisition of CDMS data, no documentation of the acquisition process existed. As a result, interviews were held with CDMS domain experts in which they started with the rules coded from the CDMS instruction document (MDC 1991), and then added control knowledge and explanation knowledge to the rules to provide a dialogue structure for the acquisition system to follow in gathering CDMS requirements from the PI/PED. (Craig 1990) describes a relationship between domain and control knowledge which we largely followed in this effort. [FIG 5] is taken from that reference.

A conventional interview technique was followed in which the knowledge engineer interviewed the domain specialist (often using the partially completed tool), encoded the acquired knowledge, and then supplied the updated system to the domain specialist for further testing [FIG 4]. (Gruber 1987) presents three principles of design for knowledge acquisition systems. Although that paper discusses automated acquisition approaches, the principles are worth noting. The first principle was that the knowledge engineer should provide a "language of task-level terms natural to the expert yet sufficient to solve the problem". In this application, knowledge of how to validate the data was already expressed to a large extent as rule structures. The second principle to be followed was that "representational primitives should be explicit and able to stand alone". Again, this posed no real problem since the existing instructions for requirements definition were already stated in an "IF-THEN" format. The third principle dealt with "avoiding generalizations except when necessary". We found that when the domain expert attempted to generalize, we often encountered new knowledge in the form of "exceptions to the rule". In this application, attempts to generalize often helped elicit knowledge.



Elicit-Encode-Test cycle
[FIG 4]

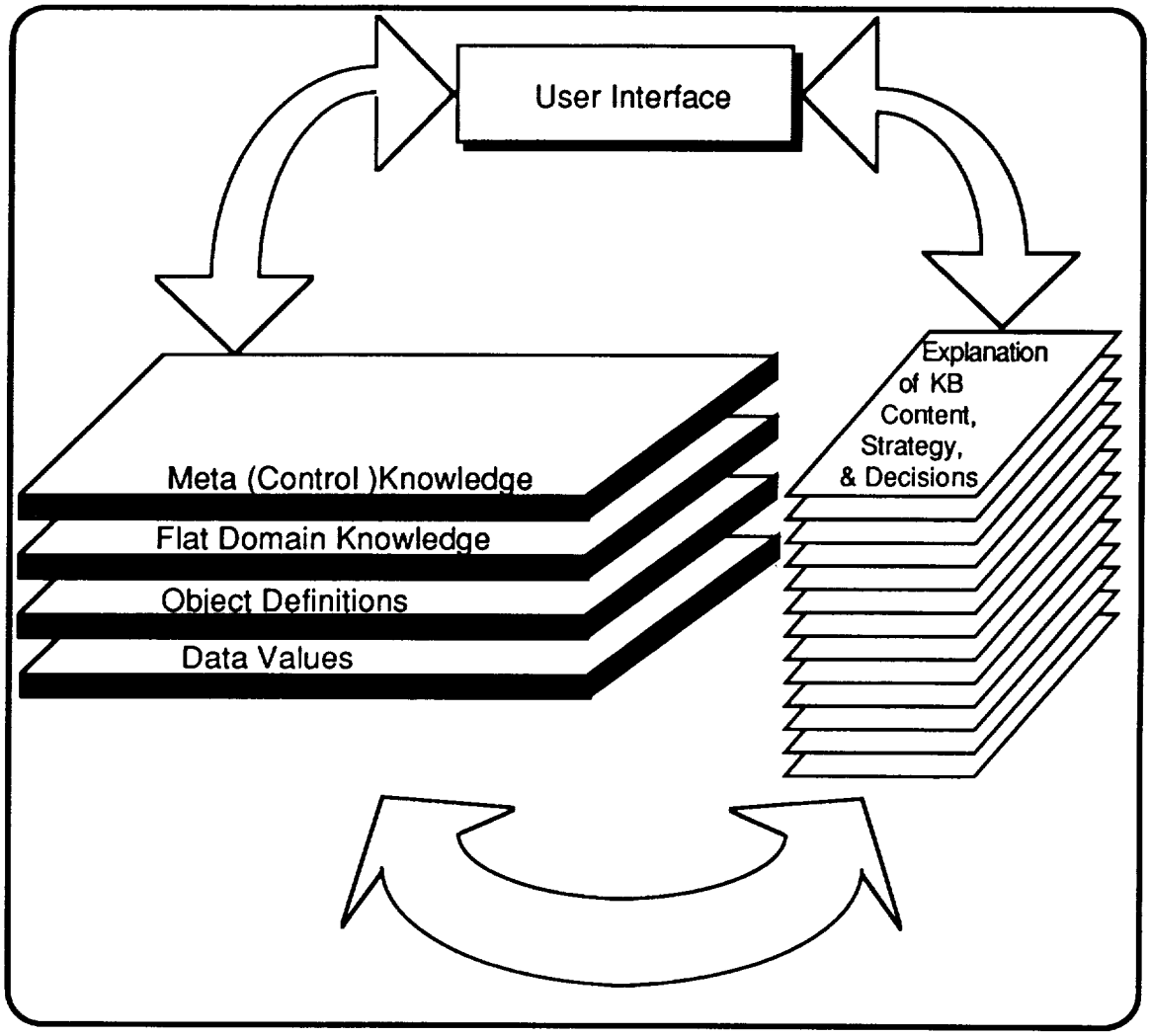
As the interviews were completed, the developer encoded the rules acquired during the interview. The tool was then made available to the domain experts for testing. One result of this incremental "ELICIT - ENCODE - TEST" development approach was that we were able to start using portions of the analysis system early in the development cycle. Another result was that customer confidence in and expertise with the tool grew during development.

2. Knowledge Representation

(Kitto 1989) points out that a failure to map properly between the Knowledge Acquisition technique, Knowledge Acquisition tool, Knowledge Representation methodology and the problem type will likely cause the effort to fail. During the domain expert interviews, most knowledge was structured as "IF-THEN" statements. This led to the use of a rule-based representation for domain knowledge, with the underlying parameters modeled as objects. The mapping between the Knowledge Acquisition technique and the Knowledge Representation paradigm was very straightforward and allowed us to model the domain naturally. In addition, the knowledge encoded in the knowledge base was easy to understand and maintain.

3. User Interface

User acceptance of a system is highly dependent on an appropriate human-computer interface. This interface must be responsive to a range of user abilities. In this effort, we have made no attempt to build user models to account for various ability levels. Our approach has been to provide information at a relatively high level, but to provide help in questions asked of the user and by providing explanations upon request. Initial experiences with the automated analysis tool indicate that this approach is sufficient. (Wexelblat 1989) gives an excellent overview of characterizations of users by ability level while (Swigger 1989) addresses research issues in human-computer interfaces for tutoring systems.



Relationship between knowledge, data and explanation

[FIG 5]

The interface to the automated acquisition tool had to provide PI/PED teams with an easy to use "point and click" interface which could provide context sensitive help when appropriate. The tool was designed to assist the PI/PED user in "constructing" CDMS requirements. The interface to the automated acquisition tool is designed to query the user for requirements definition. As the user is asked questions about parameter definitions, help is provided and constraint information is available. The queries are formed by inserting context sensitive information into a text string. For example, if a constraint existed between two data items A and B such that A has to be less than B, and if A had the value of 30, the query might look something like.. *Provide a value for B that is greater than 30*... If the user then asked for an explanation, he would be told about the relationship between A and B and the existing value for A. At that point, he could choose to supply B or modify A.

The interface to the automated analysis tool to be used by the integration team was designed to look much like the data formats the engineers were accustomed to. Less explanation is supplied and less control is exercised by the system. Both these tools were implemented in a hypertext tool (SuperCard) which allowed users to move through the data in an unconstrained fashion.

The requirements acquisition tool was somewhat more constrained than the analysis tool because it uses the dependencies relationships between the data items to guide the dialogue between the user and the system.

4. Dialogue Control

Dialogue control in the requirements acquisition tool was implemented using the dependency matrix mentioned earlier. The acquisition process begins with a leadoff series of questions, which are then used to guide or constrain further questioning. Also, much support information is provided in the query itself. This helps the user to understand the significance of the question. For example, if the user specifies that the parameter being defined requires on-board displays, the system then queries for further information on the display requirements and reminds the user that he had earlier provided the requirement for displaying the parameter. If the user indicates that the parameter will not be displayed on-board, he is not asked for unnecessary information - a situation that cannot be avoided in the paper CDMS requirements forms. This control, however, does not prevent the user from modifying earlier definitions. The details of how this is accomplished is discussed in the next section.

5. Consistency Maintenance

One of the problems with the old paper requirements approach is that users often have a need to modify requirements. As in the case given above, if a user wants to retract the requirement for on-board display of a parameter, all display requirements data for that parameter must be withdrawn. With the paper approach, relations between data across tables cannot be linked in such a way that data changed in one table changes all its associated data. In the automated acquisition system, however, these relations are modeled in a dependencies matrix so that when a datum changes, rules are triggered that modify all the associated data. This approach to consistency maintenance is loosely based in Doyle's justification-based truth maintenance system (Doyle 1981).

6. Explanation

(Craig 1990) and (Fennel 1990) point out that in many rule-based expert systems, explanation and why facilities are implemented using the rule firing chain to trace each step of the inference process. They point out that this approach is often not appropriate for providing meaningful explanations since the rule tracings often contain inferences at the wrong level of abstraction. They implemented a layered control architecture [FIG 5] in which explanations can be provided on various levels (i.e. explanation of what a datum represents, constraint knowledge about that datum, etc.) This structure was largely followed in developing the explanation system.

(Clancey 1988) identifies four categories of explanation knowledge closely related to those described above:

a. Heuristic Rules which, identify relations between data and rules using that data,
b. Structure knowledge, which identifies dependency relations among data,
c. Strategy knowledge, which identifies the procedure for applying rules, and
d. Support knowledge, which provides the justification for rules.

The dependency matrix (which identifies dependencies between data elements) is used by the explanation facility to provide justification for why a particular datum is needed. Explanation about a particular datum is statically defined either in textual or graphical format and is provided to the user upon request during a session.

IMPLEMENTATION

This Expert System is currently hosted on a Macintosh Computer using Nexpert Object for knowledge representation and inference control, Oracle for data storage, and SuperCard for the interface. The availability of commercial bridges between all these applications made interfacing them straightforward. Although some inquiries have been made as to the possibility of

rehosting the software on other platforms, no work has begun in this area. Both the knowledge base constructed in Nexpert and the data stored in Oracle will transfer to a wide range of platforms. SuperCard runs on the Macintosh platform only, but other interface tools are being considered.

FUTURE DIRECTIONS

As well as improving the existing payload integration process for Spacelab, this technology is also applicable to other aerospace applications including the Space Station Freedom program. We expect many Spacelab experiments to transition to Space Station and anticipate that having many of the experiment requirements acquired, stored in a database, and analyzed will improve the integration process within the Space Station program. Work is underway to identify which portions of the data are carrier independent.

The requirements database is expected to assist in experiment reflights aboard Spacelab where the bulk of the requirements for an experiment do not change. Tools such as these may also be used to assist mission designers in selecting payloads based on mission characteristics. For example, if a microgravity mission is being considered, designers might access the requirements database to identify candidate experiments, and then use the list of candidates to construct optimum payload configurations.

We mentioned earlier that one of the expected benefits of these systems was to use them as training tools. While the systems as currently implemented have no tutoring capabilities, engineers using them can gain a better understanding of their domain. One enhancement in future versions might be to employ tutoring strategies so that the system could be used as a teaching tool.

CONCLUSION

The PI/PED acquisition system described in this paper is planned for use on the WISP-HF/ATLAS-4 mission. WISP (a Canadian experiment) represents the first totally new experiment planned for Spacelab.

Portions of the CDMS analysis tool are currently being used by the integration team on several missions and benefits are already being realized. Maintenance of the CDMS data and the expansion of requirements definitions are much easier in the automated system, and the system has already identified a number of errors in CDMS data for missions under development.

One problem encountered during the development of the analysis tool was no access to a network server version of Oracle within TBE. All development had been done on a Macintosh using a "stand alone" Oracle. The CDMS table structures had been defined and populated with existing mission data. However; the availability of the data was limited to the single development machine and not easily accessed by other members of the integration team. A work-around solution to this problem was to use a PC database (FileMaker Pro) to store and manipulate CDMS requirements data. The expert system reads data exported from FileMaker and performs analyses and validity checks. We expect to transfer the data from FileMaker files to Oracle when the networked Oracle server hardware and software is installed.

In addition to CDMS, TBE has efforts underway to use expert Systems in several other areas of the payload integration process. One effort worth noting is the Functional Objectives Requirements Collection System (FORCS) which will be used to help PIs define functional objectives for their experiment. The current FO process suffers from many of the same problems as CDMS including the task of entering paper inputs from the PI, inconsistency in the way those requirements are stated, multiple (often inconsistent) copies of the data spread across disciplines, etc. The FO Expert System and the CDMS acquisition system are both parts of a larger effort to automate the acquisition of all experiment requirements and to store those requirements in a relational database for analysis and integration. The lessons learned in both these efforts will be applied to other

problems within the mission payload integration process.

REFERENCES

Clancey, William J.(1988). "The Knowledge Engineer as Student: Metacognitive Bases for Asking Good Questions" in Learning Issues for Intelligent Tutoring Systems, Springer Verlag

Craig. F.G., Cutts, D.E., Fennel, T.R. & Purves, B. (1990). "Graphical Explanation in an Expert System for Space Station Freedom Rack Integration", Fifth Conference on Artificial Intelligence for Space Applications, Huntsville AL, May

Doyle, John (1981). "A Truth Maintenance System" in Readings in Artificial Intelligence, pp 496-516 (Tioga Publ Co. 1981)

Fennel T.R. & Johannes, J. D. (1990). "An Architecture for Rule Based System Explanation" Fifth Conference on Artificial Intelligence for Space Applications, Huntsville AL, May 1990.

Gaines, B.R.(1989) "Integration issues in Knowledge Support Systems" in International Journal of Man-Machine Studies (1989) 31:5

Gruber, Thomas & Cohen, Paul (1987). "Principles of Design for Knowledge Acquisition" in Proceedings of 3rd IEEE Conference on AI Applications (1987)

Kitto, Catherine M. & Boose, John H. (1989). "Selecting Knowledge Acquisition Tools & Strategies based on Application Characteristics" in International Journal of Man-Machine Studies (1989) 31:8

MDC (1991). MDC G6854D Volume II, Appendix D, CDMS Forms and Instructions

Swigger, Kathleen M.(1989). "Managing Communication Knowledge" in Proceedings of the Second Intelligent Tutoring Systems Research Forum, San Antonio, TX (April 1989)

Wexelblat, Richard L. (1989). "On Interface Requirements for Expert Systems" in AI MAGAZINE 10:3 (Fall 89)

Knowledge-Based Approach for Generating Target System Specifications from a Domain Model

Hassan Gomaa, Larry Kerschberg, and Vijayan Sugumaran

Center for Software Systems Engineering
Department of Information and Software Systems Engineering
George Mason University
Fairfax, Virginia 22030-4444

Abstract

Several institutions in industry and academia are pursuing research efforts in domain modeling to address unresolved issues in software reuse. To demonstrate the concepts of domain modeling and software reuse, a prototype software engineering environment is being developed at George Mason University to support the creation of domain models and the generation of target system specifications. This prototype environment, which is application domain independent, consists of an integrated set of commercial off-the-shelf software tools and custom-developed software tools. This paper describes the knowledge-based tool that has been developed as part of the environment to generate target system specifications from a domain model.

Keywords: domain modeling, reuse, software engineering environments, object repository, requirements elicitation, knowledge-based tool support.

INTRODUCTION

An application domain is defined to be a collection of systems that share common characteristics. A domain model is used to capture common characteristics and variations among a family of software systems in a given application domain. From the domain model, a target system can be generated by tailoring the domain model according to the requirements of the target system. Thus, a target system engineer can develop the specification for a target system in terms of the domain model specified previously by a domain analyst, and does not have to perform a full systems analysis every time a new target system has to be constructed.

At George Mason University, a project is underway to support software engineering lifecycles, methods, and prototyping environments to support software reuse at the requirements and design phases of the software lifecycle, in addition to the coding phase. A reuse-oriented software lifecycle, the Evolutionary Domain Lifecycle (Gomaa, 1989; Gomaa, 1991a) has been proposed, which is a highly iterative lifecycle that takes an application domain perspective allowing the development of families of systems. A domain analysis and modeling method has also been developed (Gomaa, 1990). This paper describes a knowledge-based approach for generating target system specifications from a domain model.

DOMAIN ANALYSIS AND MODELING

The Evolutionary Domain Life Cycle (EDLC) Model (Gomaa, 1989) is a software lifecycle model that eliminates the traditional distinction between software development and maintenance. Instead, systems evolve through several iterations. Hence, systems developed using this approach need to be capable of adapting to changes in requirements during each iteration. Furthermore, because new software systems are often outgrowths of existing ones, the EDLC model takes an application domain perspective allowing the development of families of systems.

Parnas referred to a collection of systems that share common characteristics as a family of systems (Parnas, 1979). According to Parnas, it is worth considering a family of systems

when there is more to be gained by analyzing the systems collectively rather than separately, i.e., the systems have more features in common than features that distinguish them. The concept of viewing an application domain as consisting of a family of systems has been adopted by various researchers (Batory, 1989; Kang, 1990; Pyster, 1990; Lubars, 1989).

When considering the development of a family of systems, it is necessary to replace the traditional system development activities of Requirements Analysis, Requirements Specification, and System Design with activities that span the entire application domain. These are Domain Analysis, Domain Specification, and Domain Design.

A Domain Model is a problem-oriented architecture for the application domain that reflects the similarities and variations of the members of the domain (Gomaa 1992). Given a domain model of an application domain, an individual target system (one of the members of the family) is created by tailoring the domain model according to the requirements of the individual system.

A Domain Model is initially created by means of a Domain Analysis. Domain Analysis (Prieto-Diaz, 1987) is a requirements analysis of a family of systems for a domain, rather than of a given target system. A domain analysis must address the requirements of the family of current systems as well as anticipate future changes. Although some future changes may be anticipated, it is unlikely that all future changes can be anticipated. It is therefore necessary for the Domain Model to be evolutionary. It needs to be capable of evolving as new (i.e., unanticipated) requirements are added and as existing requirements are changed in unanticipated ways. Domain Analysis is comparable to a systems analysis performed on a broader scale. It involves the analysis of existing target systems in the application domain as well as interviewing domain experts and capturing their knowledge of existing features, including known or anticipated variations in the domain.

Reuse is an important goal in domain modeling. A key aspect of this work is the way it combines generation technology with composition technology. Reuse by generation (Biggerstaff, 1987) implies a top-down approach in which a target system is generated from a domain model by tailoring the domain model according to the target system requirements. Reuse by composition (Biggerstaff, 1987) is a bottom-up approach in which components residing in a reuse library are located and reused, ideally without change. Instead of requiring software developers to search large reuse libraries, the domain model has an index into the reuse library, so that reusable software components may be more easily located and included in the target system implementation.

Multiple Views of Domain Model

Applying the domain modeling method, the application domain is modeled by means of the following views:

- *Aggregation Hierarchy.* The Aggregation Hierarchy is used to decompose complex, aggregate object types into less complex object types, eventually leading to simple object types at the leaves of the hierarchy.
- *Object Communication Diagrams.* Objects in the real world are modeled as concurrent processes that communicate with each other using messages. The object communication diagrams, which are hierarchically structured, show how objects communicate with each other.
- *State Transition Diagrams.* Because each active object is modeled as a sequential process, it may be defined by means of a finite state machine and documented using a state transition diagram.
- *Generalization / Specialization Hierarchies.* As the requirements of a given object type are changed to meet the needs of a given target system, the object type may be specialized by adding, modifying, or suppressing operations. The variants of a domain object type are stored in this hierarchy.
- *Feature / Object Dependencies.* This view shows for each feature (domain requirement) the object types required to support the feature.

The domain modeling method has been applied to developing a domain model for NASA's Payload Operations Control Center (POCC) Domain.

PROTOTYPE SOFTWARE ENGINEERING ENVIRONMENT

A prototype software engineering environment is being developed, which consists of an integrated set of software tools that support domain modeling and the generation of target system specifications. A schematic representation of the prototype environment is given in Figure 1. In order to expedite development of the prototype, the environment uses commercial off-the-shelf software as well as custom software. We are using Interactive Development Environments' Software Through Pictures CASE tool to represent the multiple views of the domain model, although semantically interpreting the views according to the domain modeling method. The information in the multiple views is extracted, checked for consistency, and mapped to an underlying representation, referred to as the domain specification, which is stored in an object repository (Gomaa, 1991b).

The domain specification stored in the object repository is augmented with domain features (requirements), inter-feature dependencies and feature/object dependencies. Inter-feature dependencies capture the relationships among features. For example, a feature may require the presence of some other feature(s). Another example of inter-feature dependency is that some features may be mutually exclusive or mutually inclusive. The feature/object dependencies relate features to objects, i.e., they define the object types required to support a particular feature. The domain analyst provides this feature-related information using the Feature Object Editor. Feature/object dependencies are stored in the object repository.

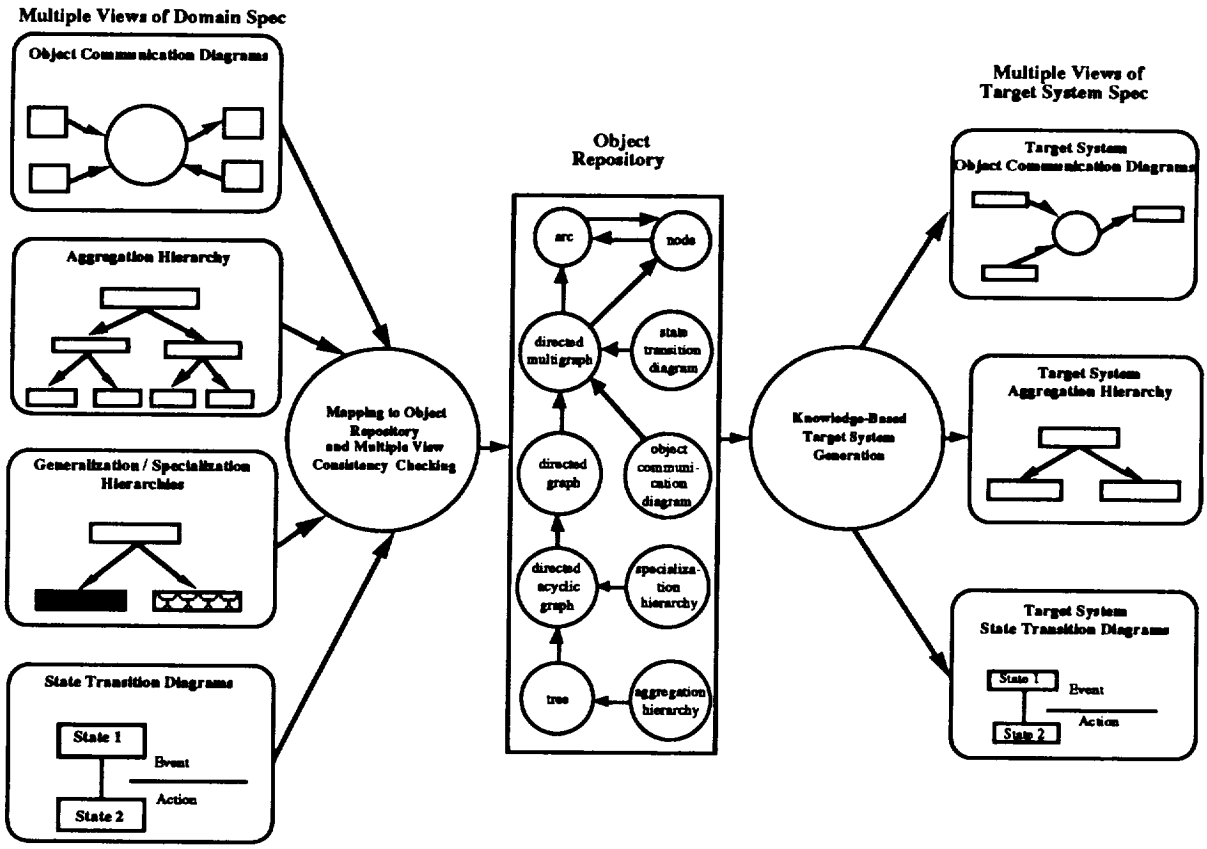


Figure 1. Prototype Domain Modeling Environment

The object repository interfaces with knowledge-based tools and provides the informal and formal specifications for reuse. Thus, the object repository provides a consistent domain model specification which can be accessed by various tools.

Once domain modeling is completed, the domain specification serves as the framework for generating target systems. The process of generating target systems from a domain model can be significantly improved with knowledge-based tool support. This tool not only must have knowledge about the domain model, but also must contain procedural knowledge about constructing target systems. A knowledge-based system called the Knowledge-Based Requirements Elicitation Tool (KBRET) is being developed to automate the process of generating the specifications for target systems. KBRET is used to assist with target system requirements elicitation and generation of the target system specification. This tool is implemented in the expert system shell CLIPS (C Language Integrated Production System), developed at NASA/Johnson Space Center (Giarratano, 1991). It conducts a dialog with the human target system requirements engineer, prompting the engineer for target system specific information. The course of the dialog is determined by the responses provided by the target system engineer. The output of this tool is used to adapt the domain model to generate the target system specification. When the target system objects have been assembled, the corresponding multiple views are derived by tailoring the multiple views of the domain model. The multiple views of the target system are then displayed using Software through Pictures.

The prototype software engineering environment is a domain-independent environment. Thus it may be used to support the development of a domain model for any application domain that has been analyzed, and to generate target system specifications from it.

KNOWLEDGE-BASED REQUIREMENTS ELICITATION TOOL (KBRET)

A target system specification is derived from the domain model by tailoring it according to the requirements specified for the target system. The process of generating a target system specification consists of gathering the requirements in terms of domain features, retrieving from the domain model the corresponding components to support those features, and reasoning about inter-feature and feature/object dependencies to ensure consistency. The Knowledge-Based Requirements Elicitation Tool (KBRET) facilitates the process of generating target system specifications from a domain model with multiple viewpoints.

The architecture of KBRET consists of two types of knowledge: domain-independent and domain-dependent knowledge. The domain-independent knowledge provides control knowledge for the various functions supported by KBRET. These functions include a browser, a feature selector, a dependency checker, and a target system generator. The domain-dependent knowledge represents the multiple views of an application domain model, including the feature/object dependencies. This knowledge is derived from the object repository through the KBRET Object Repository interface and structured as CLIPS facts (Sugumaran, 1991). The various components of KBRET are diagrammatically depicted in Figure 2.

The separation of domain-independent and domain-dependent knowledge is essential for providing scale-up and maintainability of domain specifications for large domains. Also, since the domain-independent knowledge is independent of the application domain, it can be used with domain-dependent knowledge from any application domain to generate target system specifications in that domain.

Domain Independent Knowledge

The domain-independent knowledge sources provide procedural and control knowledge for the various functions supported by KBRET. The *Dialog Manager* is responsible for carrying out a meaningful dialog with the target system engineer and eliciting the requirements for the target system. It addresses such issues as how, and in what sequence, the target system engineer should be prompted for various features, invoking and controlling the different phases of KBRET, the user interface, etc.

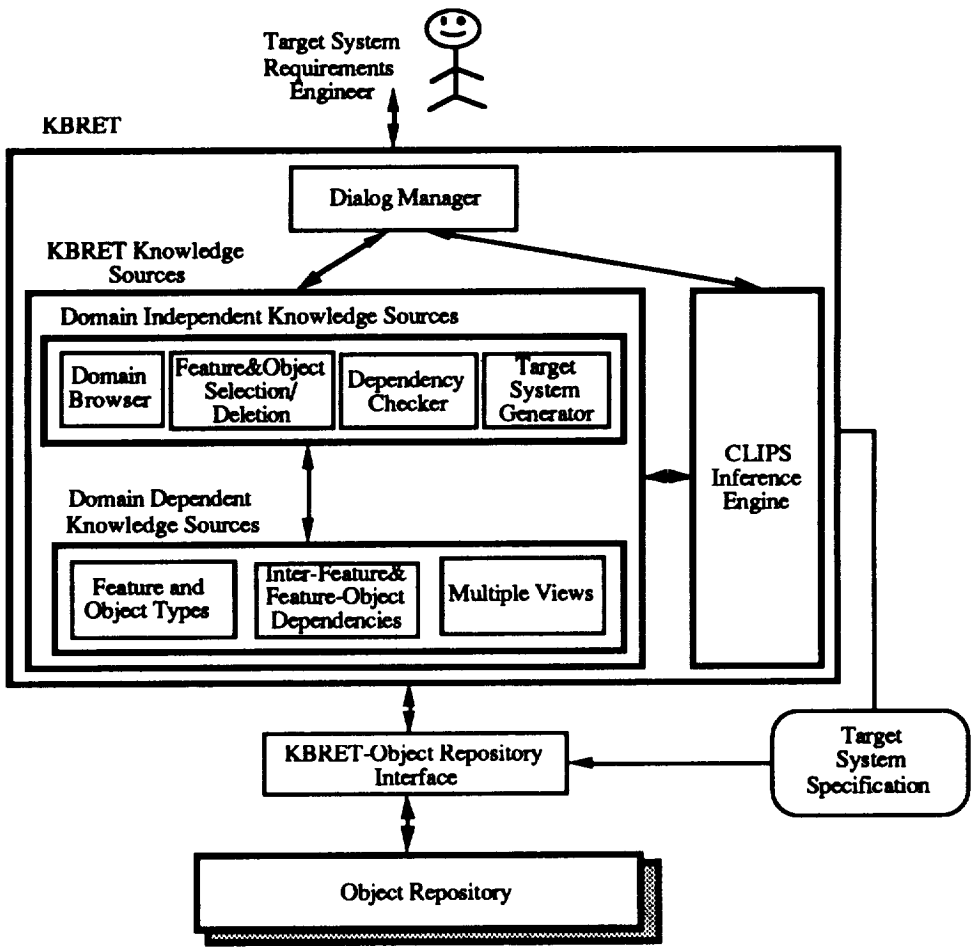


Figure 2. Knowledge Based Requirements Elicitation Tool (KBRET)

Before specifying the requirements for the target system, the target system engineer may wish to browse through portions of the domain model in order to gain understanding of the application domain under consideration. The *Domain Browser* knowledge source provides this facility. It provides rules for initiating and terminating the browsing facility and for accessing the appropriate domain-dependent knowledge sources.

The *Feature & Object Selection/Deletion* knowledge source keeps track of the selection or deletion of features for the target system and the corresponding object types. This knowledge source incorporates rules for selecting and deleting features and for invoking the appropriate rules for checking inter-feature and feature/object dependencies.

The *Dependency Checker* knowledge source cooperates with the Feature & Object Selection/Deletion knowledge source. When a particular feature is selected for the target system, the *Dependency Checker* enforces the inter-feature and feature/object dependencies for that feature. These dependencies are obtained from the *Inter-Feature & Feature- Object Dependencies* knowledge source, which is domain dependent, as shown in Figure 2. When a feature with some prerequisite features is selected, the *Dependency Checker* ensures that those prerequisite features are included in the target system. For example, in the POCC domain, the Verifying Real Time Commands feature requires the Sending Real Time Commands feature. If the Sending Real Time Commands feature is not selected and the

Verifying Real Time Commands feature is desired in the target system, the Sending Real Time Commands feature will be included in the target system before selecting the Verifying Real Time Commands feature.

Similarly, before deleting a feature from the target system, dependency checking is performed to ensure that it is not required by any other target system feature. Using the example from the previous paragraph, if both Sending Real Time Commands and Verifying Real Time Commands features are selected for the target system, the Sending Real Time Commands feature cannot be deleted from the target system as long as the Verifying Real Time Commands feature is selected for the target system. Thus, the *Dependency Checker* knowledge source has rules to enforce the inter-feature and feature/object dependencies so that a consistent target system is specified.

Once the feature selection for the target system is complete, the *Target System Generator* knowledge source begins the process of assembling the target system. The domain kernel object types are automatically included in the target system. Depending upon the features selected for the target system, the corresponding variant and optional object types are included according to the feature/object dependencies. The *Target System Generator* would detect if more than one variant (specialization) of a particular kernel or optional object type were included in the target system. These multiple variant object types have to be "integrated" to produce one integrated variant object type that would support the desired features in the target system. Some domains may require the presence of multiple variants of certain objects, and those variant objects should not be integrated. For example, in the POCC domain, multiple variants of observatory-related objects should not be integrated.

If multiple specializations of a particular kernel or optional object have been selected, and if they have to be integrated, the *Target System Generator* will access the *Multiple Views* domain-dependent knowledge source and check the appropriate generalization/specialization hierarchy to see if an integrated object type for those variant object types exists as a result of previous variant integration processes. If such an integrated object type is present, then that object type is included in the target system in lieu of those variant object types to be integrated. Once all the required integrated variant object types have been included, the target system generation is complete.

If an integrated variant object type is not in the domain model, the target system generation process is suspended until the domain analyst can specify the integrated variant and include it in the domain model. At that time the target system generation process can be reactivated to complete the specification of the target system.

Variant integration is a non-trivial task and may require considerable domain knowledge. Hence, completely automating the variant integration process will be a tremendous challenge.

Domain Dependent Knowledge

The domain-dependent knowledge sources contain specific information about a particular application domain. They are used by the domain-independent knowledge sources of KBRET in eliciting the requirements and generating the target system specification. The domain-dependent knowledge sources are derived from the domain specification, which is persistently stored in the object repository. The KBRET Object Repository Interface accesses the object repository and creates these knowledge sources using a representation that is compatible with the other knowledge sources of KBRET.

The *Features and Object Types* knowledge source contains a list of all the object types and features that have been incorporated in the domain model. For each object type, its name and properties are stored in this knowledge source. The properties of objects are: kernel, optional, variant, aggregate, agh_root, and gsh_root. The CLIPS implementation of this knowledge source is essentially a list of facts — one fact for each object type and its properties and one fact for each feature.

The various relationships and dependencies among features and between features and object types are captured in the *Inter-Feature & Feature-Object Dependencies* knowledge

source. The prerequisite relationship between two features is captured in a CLIPS fact with the key word "requires". For each feature, the object types required to support that feature are expressed as CLIPS facts using the key word "supported-by". These dependencies are enforced during feature selection or deletion by the *Dependency Checker* knowledge source.

The *Multiple Views* knowledge source contains the different views created using the EDLC methodology, in particular, the aggregation hierarchy and the generalization/ specialization hierarchies. These hierarchies are accessed and utilized by the *Target System Generator* knowledge source when the target system is being assembled. The parent-child relationship between objects in the aggregation hierarchy is expressed as CLIPS facts using the key word "is-part-of". The supertype-subtype relation between objects in the generalization/specialization hierarchy is expressed as CLIPS facts with the "is-a" key word.

GENERATION OF TARGET SYSTEM SPECIFICATION

To generate the target system specification, KBRET enters into a dialog with the target system engineer and elicits the requirements for the target system. A sample dialog, in which a domain model for the Payload Operations Control Center (POCC) application domain is used to generate a target system specification, is given in the Appendix.

At the start of the dialog, KBRET prints the system banner and asks the target system engineer whether he/she wishes to browse the domain model or would like to specify the requirements for the target system, as shown in the sample dialog in the Appendix. If the response is to browse, the browsing phase is initiated. The target system engineer can explore the domain model and get explanations for the different features incorporated in the domain model. Once sufficient familiarity with the domain model has been gained, the target system requirements specification phase may be initiated.

The target system engineer is presented with the various features captured in the domain model in the form of a menu, as shown in the Appendix, and the features desired in the target system can be selected from this menu. Whenever a feature is selected for the target system, the dependency checking phase is initiated, and the inter-feature and feature/object dependencies are checked and enforced. If a particular feature, say "F", requires the presence of other features, and they are not selected for the target system, the target system engineer is informed of that fact, and those features are automatically included in the target system in order to support feature "F".

An example of this feature dependency checking is shown in the sample dialog in the Appendix. When the target system engineer tries to select the Verifying Real Time Commands (feature 7), KBRET responds with a message saying that Verifying Real Time Commands requires Sending Real Time Commands feature and it will be automatically included in the target system, and requests the target system engineer's confirmation. When the target system engineer types "y" to confirm the selection, KBRET includes both the Sending Real Time Commands feature and the Verifying Real Time Commands feature in the target system and displays a message to that effect, as shown in the Appendix.

When a feature is selected for the target system, the object types that are required to support that feature are also selected in accordance with the feature/object dependencies and the CLIPS fact-base is updated to reflect that fact. The target system engineer, thus, can specify the requirements for the target system, and the *Feature & Object Selection/Deletion* knowledge source asserts new facts into the fact-base to record those selections. Of course, the *Dependency Checker* would ensure that the inter-feature and feature/object dependencies have not been violated.

The target system engineer can also delete features that have been selected for the target system. If a feature, say "F", is to be deleted, the *Dependency Checker* will check the fact-base to see if any of the features selected for the target system require that feature "F". If so, the deletion of feature "F" is disabled. An example of this deletion dependency checking is shown in the sample dialog. When the target system engineer tries to delete the Sending Real

Time Commands (feature 6) from the target system, KBRET responds with a message saying that the Sending Real Time Commands feature is required by the Verifying Real Time Commands feature and since Verifying Real Time Commands feature is currently selected for the target system, the Sending Real Time Commands feature cannot be deleted, and the dialog continues. When a feature "F" is deleted, it may cause the deletion of some other features if those features were included in the target system solely because of the selection of feature "F" and if they are not required by any other feature selected for the target system. The deletion of a feature also triggers the deletion of object types that were included to support that feature.

If the target system engineer would like to specify a feature that has not been captured in the domain model, the requirements elicitation phase is suspended and the domain analyst is called upon to model that requirement and enhance the domain model. Then, the target system specification and generation may be resumed.

Once the requirements for the target system have been completely specified, the target system generation phase is invoked. KBRET prompts for a name for the target system that is being generated so that it may be stored in the object repository for reuse. The fact-base is examined and the features and the object types selected for the target system are gathered. KBRET then presents the list of features that have been selected for the target system. The kernel object types are included in the target system because they must be part of every member of the family of systems. The selected variant and optional object types are examined to see if variant integration is required. If variant integration is not required, then the target system specification is generated and presented to the target system engineer.

In presenting the target system specification, KBRET provides two options. The target system engineer may view only the leaf-level object types or he/she can view both the aggregate and leaf-level object types. If the target system engineer chooses the second option, KBRET provides the aggregation hierarchy for the target system, as shown in the Appendix. This is accomplished by pruning the domain aggregation hierarchy, i.e., deleting from the domain aggregation hierarchy the object types that have not been included in the target system. KBRET presents the target system aggregation hierarchy in an indented form, as shown in the Appendix, to reflect the various levels of the aggregation hierarchy.

KBRET also outputs two files containing the target system information. These files are used to tailor the domain model graphical views and generate a set of graphical views for the target system. The target system views differ from those of the domain model in two ways. First, the optional objects that are not selected for the target system are removed. Secondly, in the case where one or more variants of a domain object type are selected, the object type is replaced by its variant(s).

If variant integration is required, the domain analyst is called upon to perform variant integration. When the integration process is completed, the target system generation phase is resumed and the target system specification is generated and presented to the target system engineer.

SUMMARY

This paper has discussed the domain modeling approach to software reuse and presented a prototype environment that supports domain modeling and generation of target system specifications. The architecture of the knowledge-based tool, KBRET, along with its domain-dependent and domain-independent knowledge sources was described. Finally, the paper has discussed KBRET's approach to generating target system specifications from the domain model, by eliciting the requirements for the target system and tailoring the domain model. A sample dialog with KBRET was also presented.

Future work includes extending KBRET to provide the capability to generate target system specifications from existing related target system specifications in conjunction with the domain model, and also to improve KBRET's user interface.

ACKNOWLEDGEMENTS

We gratefully acknowledge the assistance of S. Bailin, R. Dutilly, J. M. Moore, and W. Truszkowski in providing us with information on the POCC. We gratefully acknowledge the major contributions of Liz O'Hara-Schettino in developing the domain model of the POCC, and C. Bosch and I. Tavakoli for their major contributions to the prototype software engineering environment. This work was sponsored primarily by NASA Goddard Space Flight Center Automated Technology Branch Code 522.3 under the Code R research program, with support from the Virginia Center of Innovative Technology. The Software Through Pictures CASE tool was donated to GMU by Interactive Development Environments (IDE).

REFERENCES

Batory, D. (1989). The Genesis Database System Compiler: A Result of Domain Modeling. *Proc. Workshop on Domain Modeling for Software Eng., OOPSLA'89*, New Orleans, LA.

Biggerstaff, T., Richter, C. (1987). Reusability Framework, Assessment, and Directions. *IEEE Software*, March 1987.

Giarratano, J.C. (1991). Clips User's Guide, Version 5.0, Software Technology Branch, Lyndon B. Johnson Space Center, Houston, TX.

Gomaa, H. (1990). A Domain Analysis and Specification Method for Software Reuse. *Proc. Third Annual Workshop on Methods and Tools for Reuse*, Syracuse, NY.

Gomaa, H. (1992). An Object-Oriented Domain Analysis and Modeling Method for Software Reuse. *Proc. Hawaii International Conference on System Sciences*, Hawaii.

Gomaa, H., & Kerschberg, L. (1991a). An Evolutionary Domain Life Cycle Model for Domain Modeling and Target System Generation. *Proc. Workshop on Domain Modeling for Software Engineering, Int. Conf. on Software Engineering*, Austin, TX.

Gomaa, H., Fairley, R., and Kerschberg, L. (1989). Towards an Evolutionary Domain Life Cycle Model. *Proc. Workshop on Domain Modeling for Software Eng., OOPSLA*, New Orleans, LA.

Gomaa, H., Kerschberg, L., Bosch, C., Sugumaran, V., and Tavakoli, I. (1991b). A Prototype Software Engineering Environment for Domain Modeling and Reuse. *Proc. Fourth Annual Workshop on Methods and Tools for Reuse*. Herndon, VA.

Kang, K. C. et. al. (1990). Feature-Oriented Domain Analysis. Technical Report No. CMU/SEI-90-TR-21, Software Engineering Institute.

Lubars, M. D. (1989). Domain Analysis for Multiple Target Systems. *Proc. Workshop on Domain Modeling for Software Eng., OOPSLA'89*, New Orleans, LA.

Parnas, D. (1979). Designing Software for Ease of Extension and Contraction. *IEEE Transactions on Software Eng.*, Vol. 5 No. 2, pp. 128-137.

Prieto-Diaz, R. (1987). Domain Analysis for Reusability. *Proc. of COMPSAC'87.*

Pyster, A. (1990). The Synthesis Process for Software Development. In R. Thayer and M. Dorfman (Eds.). *System and Software Requirements Engineering*, New York:IEEE Computer Society Press.

Sugumaran, V., Gomaa, H., and Kerschberg, L. (1991). Generating Target System Specifications from a Domain Model Using CLIPS. *Proc. of Second Annual Clips Conference*, Houston, TX.

Appendix. Sample Dialog with KBRET for the POCC domain.

```
**************************************************************
*                                                            *
*  KNOWLEDGE BASED REQUIREMENTS ELICITATION TOOL             *
*                        (KBRET)                             *
*                                                            *
**************************************************************
```

Requirements Elicitation for POCC domain
****

You may browse the features incorporated in the Domain Model, specify the requirements for the Target System or quit KBRET.

Choices	Perform
1	Browse the Domain Model
2	Specify requirements for Target System
3	Quit KBRET

Please type your choice and hit return: 1

Domain Model Browsing Phase
****

Please select one of the following choices to continue.

Choices	Perform
1	Explore the Features
2	Exit Browsing Phase
3	Quit KBRET

Please type your selection and hit return: 1

Feature Exploration
****

For the description of a feature, please type its number.

Choices	Feature to be described
1	Mission Type One
2	Mission Type Two
3	Experiment Type One
4	Experiment Type Two
5	Data Collection of Simulated Telemetry
6	Sending Real Time Commands
7	Verifying Real Time Commands
e	Exit Browsing Phase

Please type your selection and hit return: 5

Data Collection of Simulated Telemetry:
****

Simulated Telemetry Data can be collected and analyzed.

Choices	Feature to be described
1	Mission Type One
2	Mission Type Two
3	Experiment Type One
4	Experiment Type Two
5	Data Collection of Simulated Telemetry
6	Sending Real Time Commands
7	Verifying Real Time Commands

```
        e       Exit Browsing Phase
Please type your selection and hit return: e

Exiting the Browsing Phase.....

You may browse the features incorporated in the Domain Model, or specify the requirements for the
Target System or quit KBRET.
    Choices         Perform
    *******         *******
        1       Browse the Domain Model
        2       Specify requirements for Target System
        3       Quit KBRET
Please type your choice and hit return: 2

          Target System Requirements Elicitation Phase
          ****************************************
Now, you will be presented with the features incorporated in the Domain Model. If a feature is desired in the
target system, please type its number and hit return. Please select one of the following choices to continue.
    Choices         Feature to be selected
    *******         **********************
        1       Mission Type One
        2       Mission Type Two
        3       Experiment Type One
        4       Experiment Type Two
        5       Data Collection of Simulated Telemetry
        6       Sending Real Time Commands
        7       Verifying Real Time Commands
        e       End selecting features
Please type your selection and hit return: 2
          The Mission Type Two Feature has been selected for the Target System.

Please select one of the following choices to continue.
    Choices         Feature to be selected
    *******         **********************
        3       Experiment Type One
        4       Experiment Type Two
        5       Data Collection of Simulated Telemetry
        6       Sending Real Time Commands
        7       Verifying Real Time Commands
        e       End selecting features
Please type your selection and hit return: 3
          The Experiment Type One Feature has been selected for the Target System.

Please select one of the following choices to continue.
    Choices         Feature to be selected
    *******         **********************
        4       Experiment Type Two
        5       Data Collection of Simulated Telemetry
        6       Sending Real Time Commands
        7       Verifying Real Time Commands
        e       End selecting features
Please type your selection and hit return: 4
          The Experiment Type Two Feature has been selected for the Target System.
```

Please select one of the following choices to continue.

Choices	Feature to be selected
*******	**********
5	Data Collection of Simulated Telemetry
6	Sending Real Time Commands
7	Verifying Real Time Commands
e	End selecting features

Please type your selection and hit return: 7

The Verifying Real Time Commands Feature requires Sending Real Time Commands Feature. The Sending Real Time Commands Feature will be automatically included if the Verifying Real Time Commands Feature is desired.

Please type 'y' to include or 'n' to not include the Sending Real Time Commands Feature. (y/n): y

The Sending Real Time Commands Feature has been selected for the Target System.
The Verifying Real Time Commands Feature has been selected for the Target System.

Please select one of the following choices to continue.

Choices	Feature to be selected
*******	**********
5	Data Collection of Simulated Telemetry
e	End selecting features

Please type your selection and hit return: e

Target System feature selection has been exited...

The following features are currently selected:

****

Number	Feature Name
*******	**********
2	Mission Type Two
3	Experiment Type One
4	Experiment Type Two
6	Sending Real Time Commands
7	Verifying Real Time Commands

Please select one of the following choices to continue.

Choices	Perform
*******	*******
1	Select more features for Target System
2	Delete a feature from Target System
3	Specify features not in the Domain Model
4	Initiate Target System Generation Phase
5	Quit KBRET

Please type your selection and hit return: 2

Please select one of the following choices to continue.

Choices	Feature to be deleted
*******	**********
2	Mission Type Two
3	Experiment Type One
4	Experiment Type Two
6	Sending Real Time Commands
7	Verifying Real Time Commands
e	End deleting features

Please type your selection and hit return: 6

Since the Sending Real Time Commands Feature is required by the Verifying Real Time Commands Feature and since the Verifying Real Time Commands Feature is currently desired in the Target System, the Sending Real Time Commands Feature may not be deleted now.

```
Please type (c) and hit return to continue: c

Please select one of the following choices to continue.
    Choices        Feature to be deleted
    *******        *********************
       2           Mission Type Two
       3           Experiment Type One
       4           Experiment Type Two
       6           Sending Real Time Commands
       7           Verifying Real Time Commands
       e           End deleting features
Please type your selection and hit return: 3
    Since Experiment Type One Feature is not required by any other target system feature, it will be deleted
    from the Target System Features.
Please type 'y' to delete or 'n' to abort the deletion of Experiment Type One Feature (y/n) : y
    The Experiment Type One Feature has been deleted from the Target System.

Please select one of the following choices to continue.
    Choices        Feature to be deleted
    *******        *********************
       2           Mission Type Two
       4           Experiment Type Two
       6           Sending Real Time Commands
       7           Verifying Real Time Commands
       e           End deleting features
Please type your selection and hit return: e

Target System feature deletion has been exited...

The following features are currently selected:
************************************
    Number         Feature Name
    *******        ************
       2           Mission Type Two
       4           Experiment Type Two
       6           Sending Real Time Commands
       7           Verifying Real Time Commands

Please select one of the following choices to continue.
    Choices          Perform
    *******          *******
       1           Select more features for Target System
       2           Delete a feature from Target System
       3           Specify features not in the Domain Model
       4           Initiate Target System Generation Phase
       5           Quit KBRET
Please type your selection and hit return: 4
    Invoking the Target System Generation Phase.....

        Target System Generation Phase:
        ***************************
Please input a name for the Target System: EXAMPLE

            EXAMPLE Target System Components
            ********************************
    The following features have been selected for the EXAMPLE Target System
    ************************************************************
                Mission Type Two Feature
```

Experiment Type Two Feature
Sending Real Time Commands Feature
Verifying Real Time Commands Feature

Assembling the EXAMPLE Target System. Please Wait.....

The Target System Object Types have been assembled. To view those object types included in the Target System, Please select one of the following choices:

Choices	Perform
*******	*******
1	View Leaf Level Object Types
2	View Aggregate and Leaf Level Object Types

Please type your selection and hit return: 2

The Aggregate and Leaf Level Objects of the Target System:
****

Payload Operations Control Center Domain (kernel aggregate)
- Telemetry (kernel aggregate)
  - Telemetry Pre-Processor (kernel)
  - Spacecraft Telemetry Processor (kernel aggregate)
    - Mission Two SC Eng. Telemetry Analog Limits Checker With Eqn. Processing (variant)
    - Mission Two SC Engineering Telemetry Trend Analyzer (variant)
    - Mission Two SC Engineering Telemetry Equation Processor (variant)
    - Mission Two Discrete SC Engineering Telemetry Analyzer (variant)
    - Mission Two FDF Interface (variant)
  - Observatory Telemetry Processor (kernel aggregate)
    - Experiment Two Instrument Telemetry Analog Limits Checker (variant)
    - Experiment Two Instrument Telemetry Trend Analyzer (variant)
    - Experiment Two Discrete Instrument Telemetry Analyzer (variant)
    - Experiment Two Scientific Telemetry Analyzer (variant)
  - TAC Interface (kernel)
  - RUPS Interface (kernel)
- Command (kernel aggregate)
  - Command Load Processor (kernel aggregate)
    - Satellite Bound Command Load Processor (kernel)
    - Earth Bound Command Load Verifier (kernel)
    - Command Load Data Store (kernel)
    - OBC Image Verifier (kernel)
    - CMS Interface (kernel)
  - Real Time Command Processor (optional aggregate)
    - Satellite Bound Real-Time Command Processor (optional)
    - Earth Bound Real-Time Command Verifier (optional)
    - Real-Time Command Data Store (optional)
  - Satellite Bound Command Problem Resolver (optional)
- Flight Operations Analyst (kernel aggregate)
  - FOA STOL Interface (kernel)
  - POCC Mode Selector (kernel)
  - FOA Command Processor (kernel)
  - FOA NCC Processor (kernel)
  - FOA Telemetry Processor (kernel)
  - NCC Interface (kernel)
- History (kernel aggregate)
  - Telemetry History (kernel)
  - Command History (kernel)
  - Flight Operations Analyst History (kernel)
  - Telemetry Block History (kernel)

The EXAMPLE Target System Generation is complete. The object types shown above have been included in it and no variant integration is required.

Combining Factual and Heuristic Knowledge in Knowledge Acquisition*

Fernando Gomez, Richard Hull, Clark Karr
Department of Computer Science
University of Central Florida
Orlando, Fl 32816
gomez@cs.ucf.edu (407) 823-2764
Bruce Hosken, William Verhagen
Grumman

Abstract

A knowledge acquisition technique that combines heuristic and factual knowledge represented as two hierarchies is described. These ideas have been applied to the construction of a knowledge acquisition interface to OPERA (Expert System Analyst). The goal of OPERA is to improve the operations support of the computer network in the space shuttle launch processing system. The knowledge acquisition bottleneck lies in gathering knowledge from human experts and transferring it to OPERA. OPERA's knowledge acquisition problem is approached as a classification problem-solving task, combining this approach with the use of factual knowledge about the domain. The interface has been implemented in a Symbolics workstation making heavy use of windows, pull-down menus, and other user-friendly devices.

1 Introduction

The goal of OPERA (Expert System Analyst; Adler, 1989) is to improve the operations support of the computer network in the space shuttle launch processing system. The checkout, control and monitor subsystem (CCMS) is a distributed computer network, which integrates software, microcode, display switches and hardware interface devices. OPERA is intended to function as a consultant to the operations staff assigned to each CCMS task. Two basic expert systems form OPERA: the Real Time System Error Manager (RTSEM) and the Problem Impact Analyst (PIA). When an error occurs, RTSEM displays information on this error obtained from a data base of errors. This information, although based on the CCMS message catalog information, contains experiential knowledge that "resides in the head of the human experts, not in texts." The knowledge acquisition bottleneck that the designers of OPERA are presently experiencing is in gathering this knowledge from the human experts and transfering it to OPERA in a form assimilable by the data structures and algorithms of the expert system. OPERA contains about one hundred thirty of these errors, but the actual number of errors in the computer network is greater than one thousand. Hence, OPERA is short in its knowledge base by a factor of ten. The goal of this project is to build a knowledge acquisition interface by means of which a domain expert without knowledge of OPERA or expert systems will be able to transfer his/her knowledge about the computer network errors to OPERA.

*This research is being funded by NASA-KSC Contract NAG-10-0058

OPERA is not a diagnostic expert system whose task is to identify or recognize a problem or error from a set of symptoms and other data. When an error occurs the computer network identifies the error with a code number. Then, OPERA's task is not one of deciding which error has taken place, but rather one of printing the pertinent information concerning that error. This information basically consists of the probable causes of the error, diagnostic advisories (actions to be performed to find out the causes of the error in case they are unclear) and the steps to be taken to correct it, called operational advisories. Table 1 depicts the information about a typical error.

Table 1. Information about a typical error.

```
  Message depicted on the firing room
  consoles

{ FEP 141 ($$$$) MICROCODE DID NOT
  RECEIVE AN ACKNOWLEDGE SIGNAL FROM
  THE I/O ADAPTER, DATA ACQUISITION
  HAS BEEN INHIBITED.  MICAS=$$$$,
  NSB=$$$$ }

{ ** TERMINAL ERROR FOR THE GSE FEP.
     THE I/O ADAPTER DID NOT SEND AN
     ACKNOWLEDGE SIGNAL TO THE
     MICROCODE DURING THE OPERATION
     INDICATED BY MICAS.

Probable cause(s):
  1.  I/O Adapter failed.
  2.  GSE Option Plane failed.
  3.  I/O Adapter port on 4-port
        controller failed.
  4.  FEP T/R failed.

Operations advisory:
  1.  Halt CPU, and record CPU
      registers. Push CPU through
      recovery.
  2.  If redundant FEP hasn't taken
      over, configure another FEP,
      or $CLAI existing FEP again.
  3.  $SPRCVE
  4.  If redundancy isn't available,
      and original FEP fails to
      $CLAI, then troubleshoot per
      following diagnostic advisory.
  5.  Lookup the MICAS in the
      microcode listings, and verify
      the operation being executed
      at the time of the anomaly.

Diagnostic advisory:
  1.  $DPLORT LI 5
  2.  SEQ FEPID1, If errors occur,
      I/O Adapter thumbin may assist
      troubleshooting.
  3.  GSE M02
  4.  SEQ FEVTR1
      (loop T/R via RCVS). }
```

When malfunctions occur, messages like this one (in the figure it is displayed in the first set of braces) appear on the firing room consoles of the system engineers monitoring launch activities. The error message designator, FEP 141, indicates the sub-system of the problem (in this case, the Front End Processor), and the error number. Dollar signs are used as place holders for actual hexadecimal addresses. This error occured because the FEP's Input/Output adapter did not send an acknowledgement to the microcode during the operation indicated by the address in the MICAS register. OPERA's response to this message is as follows (OPERA output is the text in the second set of braces). The text, denoted by two asterisks, is a note field obtained from the network system's documentation. This is provided to the system engineer as a convenience so that that he/she does not need to take the time to consult the manuals.

Probable causes for this error are listed next. Causes are listed such that the first probable cause is the most likely, the second is the second most likely, etc. More than one

problem cause may apply to the error. For this particular error, the probable cause is a failed piece of hardware; from the most specialized piece of hardware, the I/O Adapter, down to the most general, the FEP's transmit/receive circuitry.

After the probable causes, the operations advisory is listed. This set of advisories details what should be performed to remedy the situation while the launch is currently underway. Because of this requirement, any action that would jeopardize the launch can not be included in this advisory. Step 4 mandates that if a redundant FEP is not available, the potentially failing FEP is taken off-line and is given a more thorough examination using the diagnostic advisory.

The diagnostic advisory consists of a series of actual diagnostic programs to execute that may determine the cause of the problem. These procedures can not be run on any equipment that is necessary to the continued success of the launch.

However, OPERA has nothing to do with the content of this information. This has been gathered by human experts who are familiar with the computer network. Experts may disagree strongly about the content of this information, but, again, OPERA does not help the experts to gather this information, or to choose between disparaging information. Of course, the value of OPERA as a consultant to the humans who are monitoring the network depends directly on the appropriateness and correctness of the information printed by OPERA.

2 OPERA: A Classification Problem-Solving Task

At first sight, one may think that the task of building a knowledge acquisition interface for OPERA is just one of building a data entry program that will transform the English text about the errors given by the experts into the data structures of OPERA. This clearly will not affect the operation of OPERA. But if the information about the errors is incomplete or incorrect, OPERA would be of very little use to the humans monitoring the computer network. It is clear that the acquisition of the correct knowledge from the experts is essential, if OPERA is to serve a credible role as consultant.

Although OPERA has not been designed as a classification task (Gomez and Chandrasekaran, 1984; Clancey, 1985), and, as a result, there is not a hierarchy of concepts mediating the knowledge about the errors, the knowledge for each error gathered by human experts and printed by OPERA clearly constitutes a classification task. In classification problem-solving, knowledge is organized into a hierarchy of concepts. Top concepts in the hierarchy represent the most general concepts. Lower concepts in the hierarchy are refinements of the upper concepts. The main idea behind this methodology is that concepts, rather than lower level constructs such as rules or procedures, provide the criteria to analyze and organize domain knowledge and acquire knowledge from experts. This translates into the following knowledge acquisition maxim: "Do not ask a domain expert for the rules or procedures he/she uses in analyzing an error or problem, ask him/her for the concepts that he/she uses to conceptualize or classify the error, and then you can ask him/her for the rules or procedures." From a problem solving point of view, the hierarchy forces the expert to make explicit the high level conceptual steps (nodes in the hierachy) which he/she will have to consider in determining the probable causes, advisories, and diagnostic steps for a given error. From a knowledge acquisition point of view, approaching this task as a classification task becomes a necessity if the knowledge acquisition interface is going to go beyond

a data entry program, which would merely prompt the user for the probable causes, advisory, etc. The knowledge acquisition interface uses the hierarchy to automatically depict knowledge stored in the upper concepts upon request of the human expert. Then, while a human expert is adding knowledge about an error, he/she may decide to consult knowledge that he/she has stored in the upper concepts. The detailed way in which this is done is explained in section 5.

The knowledge of most domains may be divided into *factual or hard* and *heuristic or soft*. Heuristic knowledge is problem-solving knowledge about a domain. In most cases, there is no concensus among experts about how this knowledge should be organized, what constitutes this knowledge, its activation, etc. This situation is reflected in the saying: "each expert has her/his own book." The trouble shooting knowledge that diagnosticians have clearly falls within this type of knowledge. In contrast to *heuristic* knowledge, *factual* knowledge reflects the way things are. There is little disagreement among experts about what constitutes this type of knowledge. The knowledge that a pathologist has about the human body clearly falls within this category. These two types of knowledge are not dichotomous ones, but rather there is a rich interrelation between them. The heuristic problem-solving knowledge of a diagnostician may have need of the factual knowledge, especially in those cases in which the solution of a problem cannot be obtained directly by applying some right-at-hand rules.

The object of this paper, however, is not to explore the relation between problem-solving on one hand, and heuristic and factual knowledge on the other hand, but rather to investigate the relation between knowledge acquisition and these two types of knowledge. In the next two sections, we show the role that these two types of knowledge play in knowledge acquisition within the domain of the CCMS network.

3 A Factual Knowledge Hierachy for the CCMS Network

In the domain of CCMS network errors, a taxonomy of errors may be built based on the structural components of the network. This classification hierarchy is based on "hard" knowledge and does not follow any heuristic principles. It reflects the way things are. Figure 1 depicts a portion of this hierarchy. The three children of the root node, stand for Front End Processor Messages, Input/Output System Messages and Operating System Integrity Messages. The FEP Messages are in turn divided into four categories: Ground Support Equipment, Launch Data Bus, Pulse Coded Modulation and Uplink messages. These in turn are subdivided into further categories. The IOS2 submessages listed are not terminal nodes, but instead are categories that in turn are subdivided into other categories. Finally, the terminal nodes of this hierarchy will consist of individual error messages.

The relevance of this hierarchy for knowledge acquisition is that knowledge stored under the nodes of this hierarchy may be used by the human expert while she/he is in the process of adding experiential/heuristic knowledge about individual errors. The knowledge stored under these concepts are causes, advisories and corrective steps. This knowledge, as we have been reiterating, is factual and resides in the manuals describing the CCMS network. Some of this knowledge may be very relevant to a domain expert when he/she is entering the causes, advisories, etc. for a specific error. This is similar to the situation of a physician who finds it necessary to consult a medical text book about the functions of organs, while diagnosing a patient.

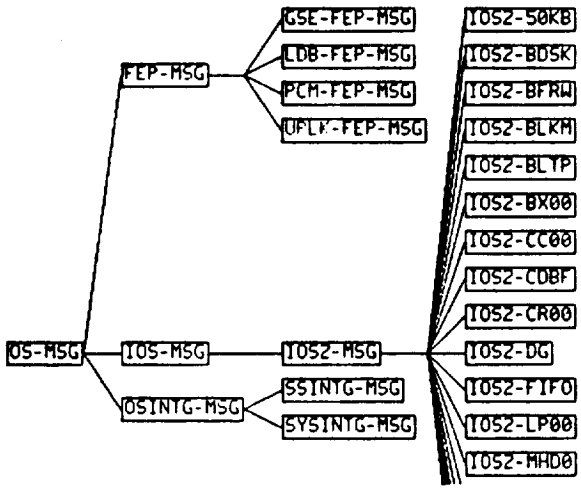


Figure 1: A Portion of the Hard Knowledge Hierarchy of the CCMS Network

The knowledge in the hierarchy is organized following strict inheritance rules. That is, every piece of knowledge in an upper-concept is true of all its subconcepts. As concepts approach the tip nodes, the knowledge becomes more specific. The user may traverse this hierarchy by using the mouse either in a top-down or in a bottom-up fashion. Or he/she may visit any concept without following any predetermined order. The knowledge will be displayed to him/her by the interface. Then, she/he may decide to consult the knowledge or use that knowledge in its entirety or partially (see section 5).

4 A Heuristic Knowledge Hierarchy for the CCMS Network

The place of the concepts in this hierarchy, and the knowledge stored under each concept, do not obey strict or hard rules; rather, they depend on the way in which a given human expert approaches the solution of a problem. As a consequence, heuristic classification hierarchies vary from expert to expert. Figure 2 depicts an elaborated heuristic hierarchy. When a domain expert starts using the Interface, he/she has at her/his disposal the factual hierarchy and an initial heuristic hierarchy similar to the one depicted in Figure 2 but much less detailed. This initial heuristic hierarchy is provided to the expert as a basis for him/her to start building his/her own hierarchy. Of course, he/she may disagree with the structure and/or content of the hierarchy, and as a consequence he/she may decide to change this initial hierarchy to conform to his/her view of the problem-solving knowledge.

In building this hierarchy, an expert is instructed to proceed in a top-down manner. The Interface walks a domain expert who is unfamiliar with the interface through the following steps:

- What are the most general categories (software, hardware, etc.) that come to your mind when the error, say, FEP-132 occurs"?
- Once you have determined that the error is, say, a software problem, which subcategories within the software do you think about?
- Which advisories and/or causes are known for a given category?

Once the domain expert has acquired some familiarity with the interface, the knowledge acquisition process concentrates on entering advisories about individual errors. During this process, the domain expert may decide to modify the heuristic hierarchy, by adding new links, altering existing links or deleting or adding advisories stored under the nodes. But, in most cases, the expert may use the knowledge stored by him/her in the heuristic hierarchy and in the factual hierarchy in order to build knowledge about individual errors. This is explained in detail in the section below.

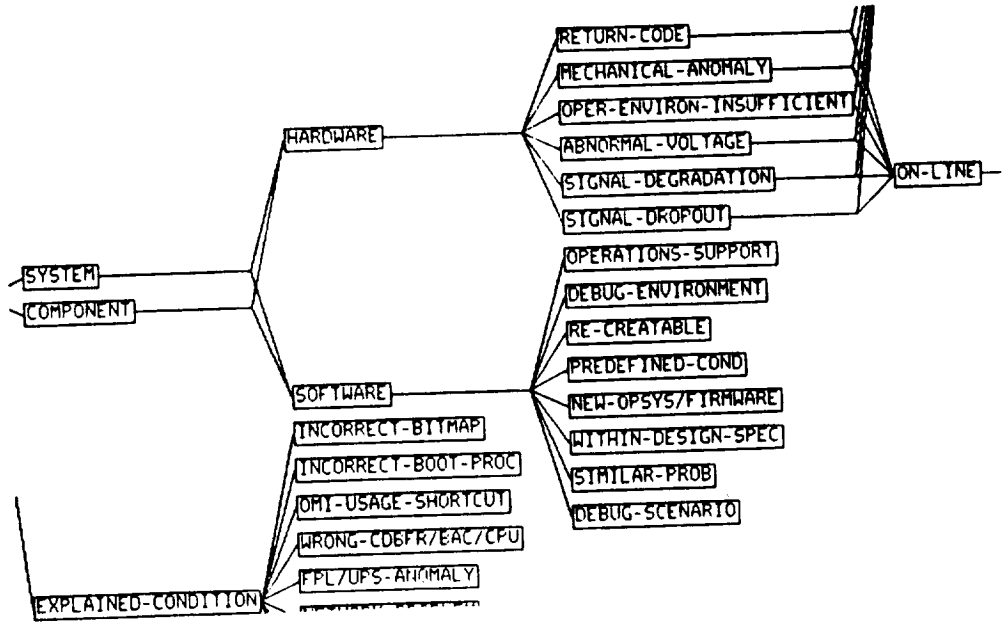


Figure 2: A Portion of an Elaborated Heuristic Hierarchy for the CCMS Network

5 A Walk Through The Interface

The interview process has two phases; the first is the construction or modification of the domain expert's error classification hierarchy, and the second is the generation of OPERA advisories. These two phases need not be strictly ordered and can be interleaved, i.e., domain experts are not forced to construct their final classification hierarchies before any advisories are created, but rather they are free to change their hierarchies at any time. To minimize the amount of startup time and to give the domain expert an idea of what we are after, we provide an elaborated error classification hierarchy designed from Grumman systems engineer Bill Verhagen's hierarchy (see Figure 2). This hierarchy provides systems engineers unfamiliar with the interface a starting point from which they can begin to coalesce their experiential knowledge of the CCMS network. While initial interviews require some instruction and typically last several hours, a given interview session can be accomplished in as little as 30 minutes, depending on the amount of information to be elicited.

5.1 Creating and Editing OPERA Advisories

The primary goal of the interface is the acquisition of knowledge about error messages. Currently the data collected are exported to the OPERA system in the form of advisories enumerating the probable causes, operational advisories, and diagnostic advisories for specific errors generated by the CCMS network. The first step in creating an advisory is choosing the error message to describe. The user is presented a menu of error messages that were previously specified by the Knowledge Engineer. The error messages on this menu reflect those errors that the Knowledge Engineer is in-

terested in collecting information about. The user is free to choose any message on the menu.

5.1.1 Placing Errors in the Heuristic Hierarchy

Once an error message has been chosen, the user is asked to place the error within his current heuristic hierarchy. To aid the user in this task, the interface provides help in the form of status register decodings and notes provided by the Knowledge Engineer.

Given this help, the user should be able to place the error in his heuristic hierarchy. Placing the error within the heuristic hierarchy is a matter of specifying which node is to become the error's parent. If a suitable parent does not exist in the hierarchy, the user is given a chance to create and place the parent in the hierarchy at that time. It may be, however, that the parent of the parent (grandparent of the original error message) does not exist in the hierarchy. Again, the user may create and place the grandfather in the hierarchy. This process can continue as long as necessary until the chain of new error categories can be linked to a node in the hierarchy (see Figure 3). Once the error has been inserted into the hierarchy, the interface gives the domain expert the opportunity to create the list of probable causes, operational advisories, and diagnostic advisories associated with the error.

5.1.2 Causes, Operational Advisories, and Diagnostic Advisories

Adding and editing cause and advisory information is quite simple. A pop-up menu is presented that allows the user to pick between changing probable causes, operational advisories, or diagnostic advisories. Once an area has been selected the interface allows the user to: add new lines of information, edit specific lines, rearrange the order of lines, or delete lines. Each line consists of free-form text keyed in by the user or mouse-selected from default information contained in the factual and heuristic hierarchies. Figure 4 shows the interface screen during the entry of probable cause data for the FEP 132 error.

5.1.3 Using the Default Information

As mentioned above, the user may create advisories by selecting text, via the mouse, from the factual and heuristic hierarchies. The texts available to be selected are those default advisories constructed by the domain expert and knowledge engineer and stored in the heuristic and factual hierarchies. When the user is to the point of entering in a line of text of the probable causes, operational advisory, or diagnostic advisory, the system displays the default advisories in the lower right window pane of the interface screen (see Figure 5). How the interface determines which default advisories are displayed in this pane is described below.

First, the interface must determine whether the user has chosen to display information from the factual hierarchy, from his own heuristic hierarchy, or both. This determination is based on the option the user has chosen using the *Select Inheritance* command (the default option is to show both). If the user has chosen to display both or has simply taken the default, the interface will collect default advisories from both hierarchies, displaying the user's own defaults at the top of the window. This is done under the assumption that the expert will feel that his own default advisories are more relevant than those of the knowledge engineer. If the user chooses one or the other type of knowledge, the interface will collect only the default advisories from the corresponding hierarchy.

Given that the system knows which hierarchy or hierachies to collect the default advisories from, the interface then uses the hierarchy's structure to decide which advisories to display. For example, suppose the FEP 132

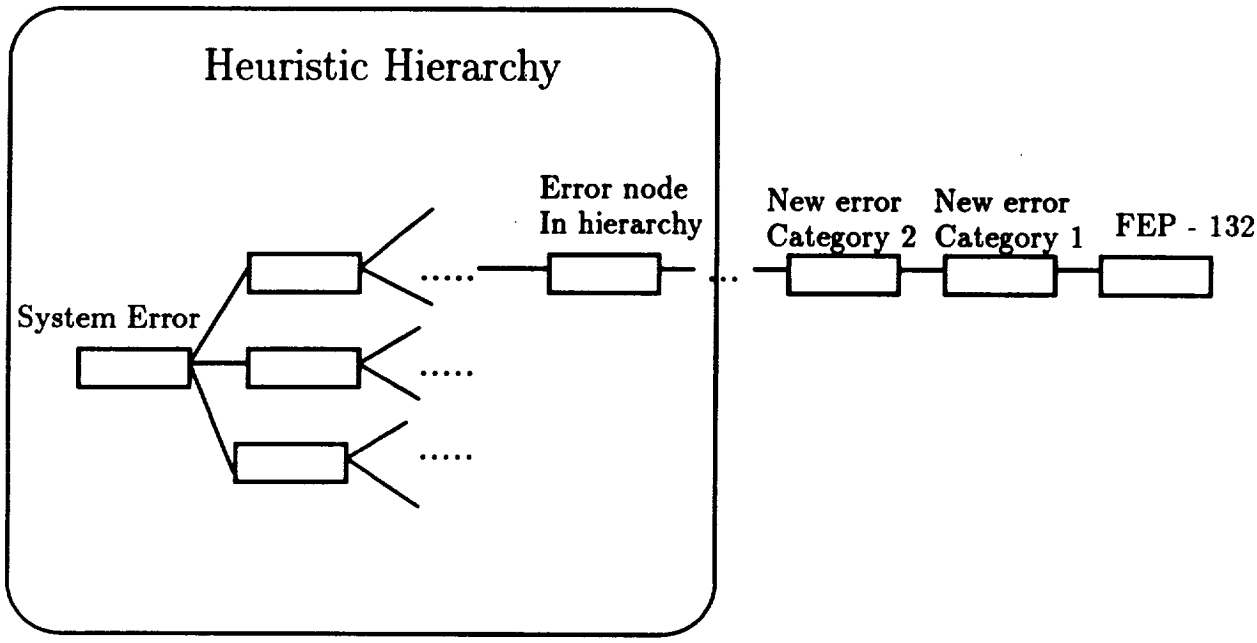


Figure 3: Adding Error Message to the Heuristic Hierarchy

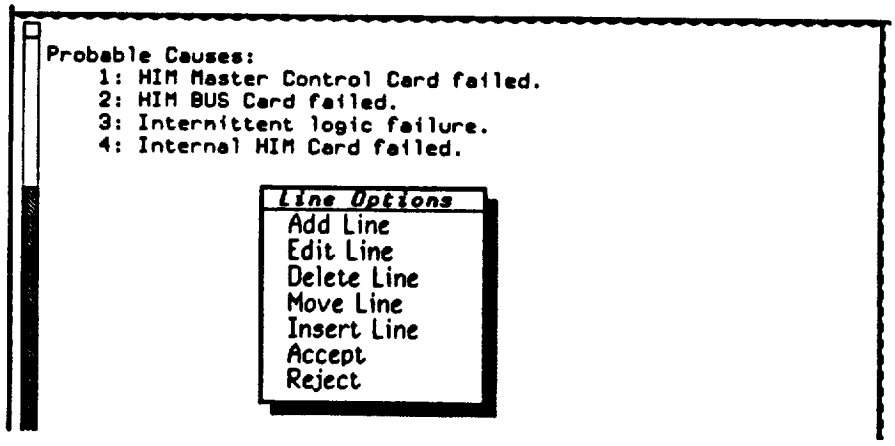


Figure 4: Entering Probable Cause Information

error used above was classified as a mechanical anomaly and the user had chosen to use default advisories from his own hierarchy. The interface would begin collecting default advisories from the mechanical anomaly node in the hierarchy. These advisories are the most specific and will be displayed at the top of the window pane. The interface then traverses up the hierarchy to the ancestors of the mechanical anomaly node. The default advisories for each ancestor are collected and added to the list of advisories to be displayed after the advisories found in mechanical anomaly. This process continues until the root node is reached along each ancestral path. The by-product of this process is a list of all the advisories from the parent of the error we are describing up to the root of the hierarchy in order from most specific to most generic.

Once the advisories have been collected, the user can select them using the mouse and include them, as is, in his description, or modify them in anyway he chooses. This means that the domain expert does not need to store "perfect" advisories, but can store advisory templates that can be modified as necessary. This greatly enhances the flexibility of the interface.

Another enhancement stems from the fact that the default advisories themselves are stored in a hierarchical structure. This allows different levels of default information to exist and be used by the domain expert. One example we encountered where this was useful was in the specification of diagnostic advisories. Typically, a diagnostic advisory includes reference to sequences of diagnostic programs that should be executed. The case may be, however, that an entire diagnostic sequence need not be run but only several of its sub-tests. To accommodate this situation the domain expert may specify that a default advisory has its own children. This allows the system to recognize that an advisory detailing a sequence of tests may have children that are the actual sub-tests. For example, the diagnostic sequence SEQ CP1 - CPU DIAGNOSTIC PART 1, has the following nine sub-tests:

```
"TST02 - XORB TEST"
"TST03 - REGISTER ADDRESSING TEST"
"TST04 - R2 DATA INTEGRITY TEST"
"TST05 - BLM/BRX TEST"
"TST06 - R3-R15 DATA INTEGRITY TEST"
"TST07 - ABRB TEST"
"TST08 - NOP TEST"
"TST09 - LDX TEST"
"TST10 - IBR TEST"
```

The user can chose the string "SEQ CP1 - CPU DIAGNOSTIC PART 1" to include in his advisory by clicking the left button of the mouse when the mouse cursor is above this text, or he can see the associated sub-tests by clicking the right button. If there were sub-sub-tests, these could be viewed by clicking the right button again. Returning to a higher level is accomplished by clicking the middle button of the mouse. In summary, clicking the left mouse button selects the text under the mouse cursor, clicking the middle button takes the user up one level in the advisory hierarchy, and the right button takes the user down one level in the advisory hierarchy.

When the user has finished entering his description of an error, he may choose to save it in his hierarchy or simply abort. Saving the information amounts to creating the necessary frames and their fillers in the expert's hierarchy. If the user does not abort, the expert's hierarchy is redisplayed with the new error node included. This concludes the discussion of the error description process.

5.2 Modifying the Structure of Heuristic Hierarchies

Maintaining the domain expert's error classification hierarchy is one of the most important tasks of the interface. Several powerful options have been implemented to allow the

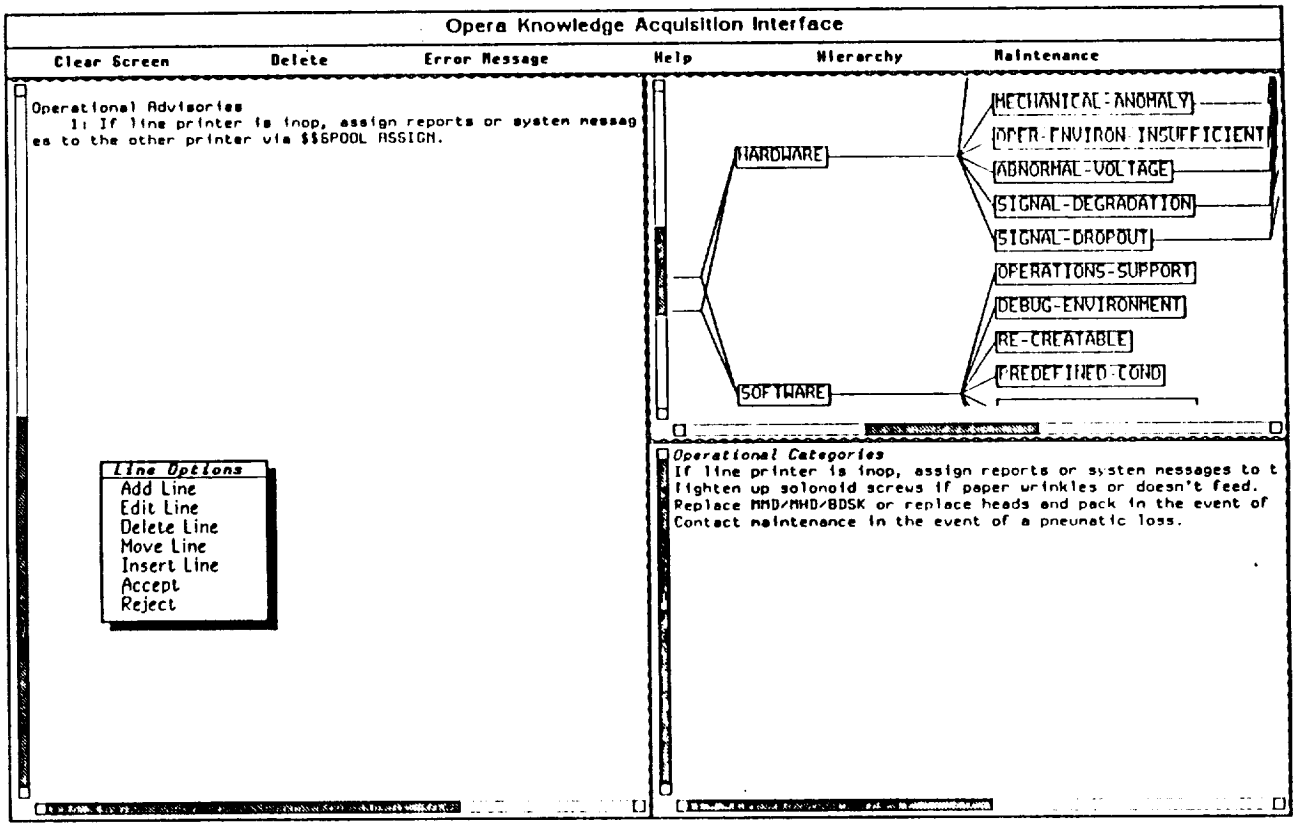


Figure 5: Interface Main Screen with Default Advisories

user to quickly and easily change the structure of his/her hierarchy. These options include adding new error categories, adding and deleting links between errors or error categories, and moving sub-hierarchies from one place in the hierarchy to another.

5.2.1 Adding New Error Categories

New error categories are added to the domain expert's error classification hierarchy using the *Add Error Category* option. This option allows the user to create a new error category and place it in the hierarchy. The user is prompted for the name of the new category and the category that is to be its parent. The parent category must exist in the hierarchy and may be given either by typing its name via the keyboard or by clicking on its graphical representation using the mouse (all the nodes in the domain expert's hierarchy are mouse selectable). After this information is given, the interface redisplays the hierarchy reflecting the addition of the new category.

5.2.2 Adding and Deleting Links

Links or inheritance paths can be added to and deleted from the expert's hierarchy using the *Add New Link* and *Delete Link* options. In the case of adding a new link, the user is prompted for a child category and a parent category. The system checks to see if the parent node is a descendant of the child node, and if it is, the attempt is aborted. This constraint guarantees that cycles will not be created by adding new links. Deleting a link is similar to adding a new one. The user is prompted for the parent and child nodes that define the end-points of the link. Assuming that the parent and child nodes given indicate an existing link, the system proceeds to remove the link and redisplay the hierarchy. Because the error classification hierarchy is a tangled hier-

archy, child nodes may have multiple parents and deleting any one of them does not effect the child node. Deleting the last link between the child and the rest of the hierarchy, however, effectively removes the child and any of its descendents that are not attached to the rest of the hierarchy through their own links (see Figure 6).

5.3 The Restructure Hierarchy Option

Should the user wish to radically restructure his/her heuristic hierarchy, the *Restructure* option can be used. This option allows the user to move sub-hierarchies from one parent node to another. The user is prompted for the root node of the sub-hierarchy he would like to move and its new parent. If the root node has several existing parents, a menu containing the names of these parents is displayed and the user is expected to click on the name of the parent node that he wishes to break away from. Constraints involving the creation of cycles and validity of node names are enforced to prevent corruption of the hierarchy. When the constraint checks are passed the hierarchy is redisplayed.

5.4 Modifying Default Information Within Heuristic Hierarchies

Information stored in the interior nodes (error category nodes) of the heuristic hierarchy is modified using the *Edit Category Data* option. With this option, users can explain the reasoning behind their classifications and create or edit default operational and diagnostic advisories. Default advisories containing the domain expert's experiential knowledge are displayed and used during the creation of OPERA advisories. Each default advisory and the expert's reasoning about his classification consists of one or more lines of text.

5.5 Specific Tools For OPERA

A special maintenance menu is provided to the knowledge engineer so that he can: add new errors to be described to the system, dump the collected advisories in a format readable by OPERA, and change the structure of the factual hierarchy. To add a new error, the knowledge engineer must enter the information that the domain expert is going to need before he can describe the error. This includes the status register values of any register inserts, the actual text of the error message, the formats of the register inserts, the notes from the CCMS documentation about this error, and the placement of the error within the factual hierarchy.

Dumping the collected advisories to OPERA is done by simply clicking a menu option. The user is then asked for the filename of the dump file. The data output is in a pseudo-LISP form that OPERA can directly input. Data may be dumped at any time and as many times as needed. Changing the structure of the factual hierarchy is handled similarly to changing the structure of the expert's hierarchy. The knowledge engineer uses the same restructuring commands that are available to the domain expert for changing heuristic hierarchies.

6 Design of the Two Hierarchies

The basic unit of information in our representation is a frame representing a single node within a hierarchy. A node (frame) may represent a root, a leaf, or an internal node within a hierarchy. Each node is known by a "node name" that is specified as an ASCII string (without spaces) by the expert creating the node. Associated with each node are two types of information: first, the information that details the hierarchy (i.e. the expert) to which the node belongs, its parent

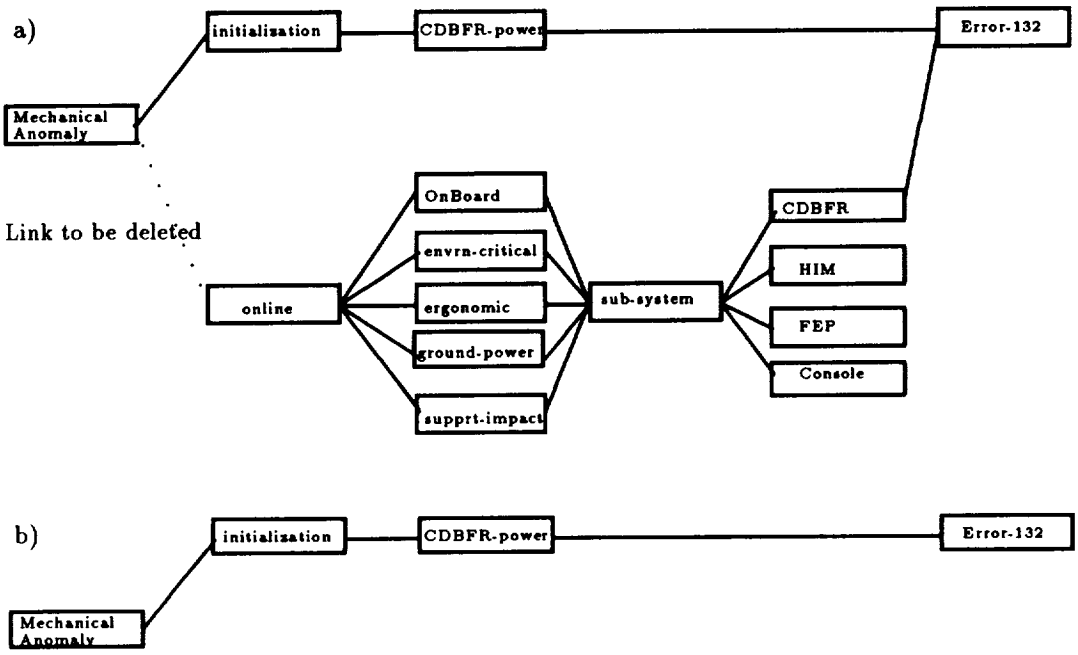


Figure 6: Deleting Links Between Nodes in the Heuristic Hierarchy: a) before, b) after

nodes, and its child nodes within the hierarchy; and second, the domain information that node stores within that hierarchy.

The frame structure for specifying nodes is as follows. A node is identified by a "name". The information detailing a node's position and connections within a hierarchy are stored under the property "*inherit*" while the domain information is stored under the property "*frame*". Within each property, the top level slot names the expert creating the node. This expert name uniquely identifies the hierarchy to which the node belongs. Within the "*inherit*" property, under the expert name are two slots, "children" and "parents", that identify the links within this expert's hierarchy. In frame notation, a node is defined as:

```
( <node-name>
  ( *frame*
    ( <expert-name1>
      ( <domain-data  ...))
    ( <expert-name2>
      ( <domain-data  ...))
              . . .
    ( <expert-nameN>
      ( <domain-data  ...)))
  ( *inherit*
    ( <expert-name1>
      ( children ( <node-name> ...))
      ( parent ( <node-name>)))
    ( <expert-name2>
      ( children ( <node-name> ...))
      ( parent ( <node-name>)))
          . . .
    ( <expert-nameN>
      ( children ( <node-name> ...))
      ( parent ( <node-name>)))))
```

The OPERA Interface is built upon a variety of primitive functions that control access to information within the entire data structure. An expert is limited to his/her hierarchy and the factual hierarchy defined by the knowledge engineer. The system's primitives control the inheritance of information within an expert's hierarchy and from the factual hierarchy to the expert's hierarchy. An expert is unaware that the data structure (frame) storing his information also stores other experts' information. Duplicate names for internal nodes

within heuristic hierarchies create no problems for keeping the domain experts' information separate.

In the OPERA domain, heuristic hierarchies share leaf nodes describing individual errors. All information entered about an error by any number of experts is recorded within the one frame describing the individual error. Contradictory and conflicting information among experts is kept segregated within each expert's subframe. In this fashion, the knowledge structure supports multiple, conflicting views of the domain without destroying the integrity of any expert's information.

A priori knowledge about the domain is stored in a hierarchy with the expert name: "FACTUAL". The system's primitives recognize "FACTUAL" as identifying the factual hierarchy. Information within the factual hierarchy is available to domain experts as they define their hierarchies and enter specific information about individual errors. The system uses the factual hierarchy to display suggestions and/or possible text for the expert to consider, modify, and incorporate in his/her hierarchies. The system prevents experts from altering the factual hierarchy.

This knowledge structure with its primitives allows multiple experts to define heuristic hierachies (which can be tangled) reflecting their view of the domain, to interact with an *a priori* knowledge base without contaminating it, and to enter information into a single data representation without fear of corrupting information entered by other experts. At the same time, all information is available to the knowledge engineer in a consolidated form requiring little manipulation to make sense of the information.

7 Conclusions and Future Research

A knowledge acquisition framework that makes use of factual and heuristic knowledge has been described. This technique has been applied to the acquisition of advisories and probable causes about errors that occur in the computer network controlling the space shuttle launch processing system. The knowledge acquisition interface is currently running on a Symbolics 3653 under version 8.1 of the Genera operating system. The implementation is in the process of being converted to run under CLIM (Common Lisp Interface Manager) in Allegro Common Lisp on a SUN platform. SUN workstations are much more common at the Space Center than Symbolics machines, and this migration should provide systems engineers with greater accessibility to the interface. Information about approximately 50 error messages has already been collected from 7 experts. While these error messages are primarily concerned with the Front End Processing sub-system, we are expanding our efforts to recruit experts with knowledge about the other sub-system messages.

Although we have applied these ideas to the construction of a knowledge acquisition interface for OPERA, and some of the components of the interface are OPERA dependent, (e.g., the final dumping of the advisories into OPERA data structures), the interface has a range of application that goes beyond OPERA. In principle, any domain that can be analyzed into a factual and a heuristic hierarchy as described in the body of the paper falls within the scope of the interface. Of course, this description is very general and some domains are going to have idiosyncracies that will require special mechanisms to handle them. However, if one stays within the area of determining the probable causes and advisories of computer network errors, then the interface can be used in many subdomains with very

minor modifications.

Table 2. A Portion of the Data Dumped From the Interface to OPERA.

```
** MSG-CAUSES: **
((FILTERS) 1.  GSE Option Plane has
               failed)
((FILTERS) 2.  GSE FEP Option Plane
               microcode has failed)
((FILTERS) 3.  GSE FEP 4-port con-
               troller has failed)
((FILTERS) 4.  GSE FEP CPU failed)

** DIAGNOSTIC-ADVISORY: **
((FILTERS) 1.  M02 on the data
               acquisition plane)
((FILTERS) 2.  SEQ FEPI01)
((FILTERS) 3.  SEQ CP1, CPU Diag-
               nostic Part 1)
((FILTERS) 4.  SEQ CP2, CPU Diag-
               nostic Part 2)
((FILTERS) 5.  SEQ OPD, Option Plane
               Diagnostic)
((FILTERS) 6. $DPLORT LI 4)
((FILTERS) 7. $DPLORT LI 5)

** INSERT-FORMAT: **
(INSERT1 ASCII CPU-NAME-INTERPRET
 CPU-NAME)
(INSERT2 HEX MDT-CDT-PTR-DECODE
 MDT-CDT-PTR)

** OPS-ADVISORY: **
((FILTERS) 1.  Note the MDT/CDT
               Pointer Address in
               the error message)
((FILTERS) 2.  If ACTIVE GSE FEP,
               verify that STANDBY
               GSE FEP is O.K.)
((FILTERS) 3.  Halt the CPU and
               record CPU regs)
((FILTERS) 4.  Perform applicable
               data retrieval progs
               $SPRCVE, $SPBLOK,
               $SPSNPR)

** MSG-TEXT: **
  FEP 142
  INSERT1
  MICROCODE DETECTED INVALID
  MEASUREMENT/COMMAND TYPE CODE,
  DATA ACQUISITION INHIBITED,
  MDT/CDT PTR =
  INSERT2
```

We are planning to incorporate in the interface some of the ideas described in (Gomez and Segami, 1990; Gomez and Segami, 1991). We are targetting two possible applications of these ideas. One is the construction of the factual hierarchy from natural language input. The other is to use natural language combined with some elicitation techniques (Boose and Bradshaw, 1987) to build the heuristic hierarchy during the first stages of its construction by domain experts unfamiliar with the interface. The final result will be the construction of a generic knowledge acquisition interface incorporating the automatic construction of hierachies from natural language input.

References

Adler, R., Heard, A., & Hosken, B. (1989). An Expert Operations Analyst (OPERA) for a Distributed Computer Network. *AI Systems In Government (AISIG)*. Washington D.C.

Boose, J. & Bradshaw, J. (1987). Expertise transfer and complex problems: using AQUINAS as a knowledge acquisition workbench for expert systems. *International Journal of Man-Machine Studies*, 26, 3-28.

Clancey, W.J. (1985). Heuristic Classification, *Artificial Intelligence*, 27, 289-350.

Gomez, F. & Chandrasekaran, B. (1984). Knowledge organization, and distribution for medical diagnosis. In W. Clancey

& E. Shortliffe, Eds. *Readings in Medical Artificial Intelligence.* Reading, MA: Addison-Wesley.

Gomez, F. & Segami, C. (1990). Knowledge acquisition from natural language for expert systems based on classification problem-solving methods. *Knowledge Acquisition*, 2, 107-128.

Gomez, F. & Segami, C. (1991). Classification Based Reasoning. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3), 644-659.

Information Management

A Spatial Data Handling System for Retrieval of Images by Unrestricted Regions of User Interest

Erik Dorfman
Hughes STX
NASA/Goddard Space Flight Center
Greenbelt, MD
dorfman@nails.gsfc.nasa.gov

Robert F. Cromp
Code 934
NASA/Goddard Space Flight Center
Greenbelt, MD
cromp@nails.gsfc.nasa.gov

Abstract

The Intelligent Data Management (IDM) project at NASA/GSFC has prototyped an Intelligent Information Fusion System (IIFS), which automatically ingests metadata from remote sensor observations into a large catalog which is directly queryable by end-users. The greatest challenge in the implementation of this catalog has been supporting spatially-driven searches, where the user has a possibly complex region of interest and wishes to recover those images that overlap all or simply a part of that region.

A novel spatial data management system is described, which is capable of storing and retrieving records of image data regardless of their source. This system has been designed and implemented as part of the IIFS catalog. A new data structure, called a *hypercylinder*, is central to the design. The hypercylinder is specifically tailored for data distributed over the surface of a sphere, such as satellite observations of the Earth or space. Operations on the hypercylinder are regulated by two expert systems. The first governs the ingest of new metadata records, and maintains the efficiency of the data structure as it grows. The second translates, plans, and executes users' spatial queries, performing incremental optimization as partial query results are returned.

1 Introduction

1.1 Needs of the scientific community

With the planned launching of the Earth Observing System (EOS) platforms and with the continuing generation of data by existing missions such as the Hubble Space Telescope (HST), NASA faces one of its greatest challenges yet: the cataloging of remote-sensor data in a manner that will allow users from a variety of scientific disciplines to quickly recover datasets of interest from the vast, constantly-expanding archive.

The ability to query or browse large catalogs of image data by the spatial characteristics of desired datasets is involved in solving what is referred to as the *spatial data handling* problem. Whether such a catalog contains downward-looking images of the Earth or outward-looking images of space, the spatial data structures resident in the catalog must support two basic spatial search operations required by the general scientific community (see Figure 1):

- *Window query*: given a region of interest, find all images that *overlap* the region.
- *Containment query*: given a region of interest, find all images that *completely contain* the region.

There is also a simple case of these queries, whose use is sometimes convenient:

- *Point query*: given a point of interest, find all images that overlap the point.

In addition, users require the ability to combine the above operations into more complex spatial queries via the operators AND, OR, and NOT.

1.2 Problems with existing approaches

Most attempts at spatial data handling in data catalogs encounter major difficulties from the start because the catalogs are implemented using relational database (RDB) packages. RDBs generally do not support data structures for handling anything other than linearly-ordered records. The object-oriented database (OODB) research of recent years provides a means of implementing spatial data structures directly inside data catalogs, and

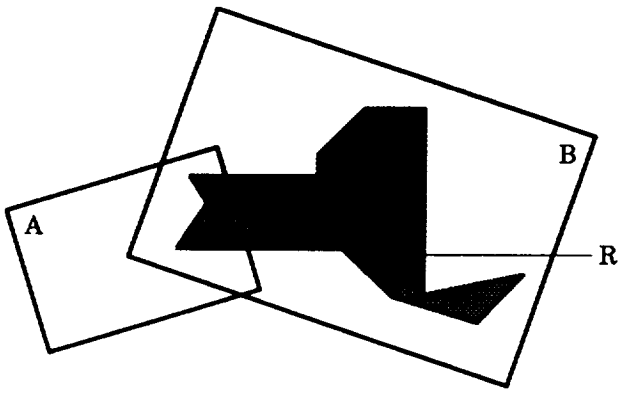


Figure 1: Images A and B both satisfy a window query on the shaded region R, but only image B satisfies a containment query on R.

OODB technology has thus been utilized in implementing the spatial data management system described in this paper.

Some catalogs circumvent most spatial data handling problems by virtue of only having to deal with queries involving a single instrument. By using information about the orbit of the instrument's platform, spatial queries are mathematically converted into sets of path-row coordinates that specify images satisfying the query, and these coordinates are used as the search keys for the images. The problem with this approach is the lack of both extensibility and flexibility. First, metadata from new platforms and instruments cannot be added without simultaneously authoring new spatial-search software. Second, images with identical path-row coordinates might not have identical locations due to fluctuations in the orbit of the platform, so a path-row-based spatial query system may falsely accept or reject images during a query.

Even catalogs that employ robust spatial data handling techniques encounter difficulties because they actually treat the globe not as a sphere but as a *planar surface*, a consequence of employing spatial data structures that use latitude-longitude based coordinate systems. The problem is that the surface of a sphere cannot be mapped onto a plane without introducing discontinuities and considerable distortion near the poles, as is evident in most cartographic projections. When using planar spatial data structures (such as quadtrees or k-d trees) to represent an inherently spherical domain, these anomalies present major difficulties in query processing and often result in an "unbalancing" of the data structure, leading to an overall drop in performance during search.

1.3 The application

The Intelligent Data Management group is conducting research into the development of data management systems that can handle the archiving and querying of data produced by Earth and space missions. Several unique challenges drive the design of these systems, including the volume of the data, the use and interpretation of the data's temporal, spatial, and spectral components, the size of the userbase, and the desire for fast response times.

The IDM group has developed an Intelligent Information Fusion System (IIFS) for testing approaches to handling the archiving and querying of terabyte-sized spatial databases (see Figure 2). Major components of the system are the mass storage and its interactions with the rest of the system [Camp91]; the real-time planning and scheduling for processing the data [Short91]; the extraction of metadata and subsequent construction of fast indices for organizing the data along various search dimensions [Camp89] [Cromp91] [Dorf91]; and the overall user interface.

The IIFS design is novel in a number of areas. Semantic data-modeling techniques are used to organize the mass storage system to reduce the transfer times of the data to on-line devices and the mechanical motions of the supporting robotics. Data percolates from near-line mass storage to on-line disk storage based upon its frequency of use. A combination of neural networks and expert systems defines how metadata is extracted to build up search indices to the underlying database. The metadata itself is organized in an object-oriented database which has special data structures for representing the multiple views of the data (such as temporal, spatial, spectral, project, sensor) without resorting to multiple copies of information. A special data structure that maps directly between the Earth and a sphere organizes the data for efficient spatial querying. The user interface is configured dynamically at run-time depending on the scientist's discipline and the current knowledge in the object database.

Experimentation with the IIFS design and implementation have shown that greater flexibility is needed in the spatial data handling routines so that images with a variety of coverage and orientation can be uniformly retrieved with respect to a user's region of interest. The remainder of the paper discusses the enhancements that have been made to the IIFS spatial data structures and describes an overall spatial data handling system that combines declarative and procedural knowledge for efficiently managing spatial queries.

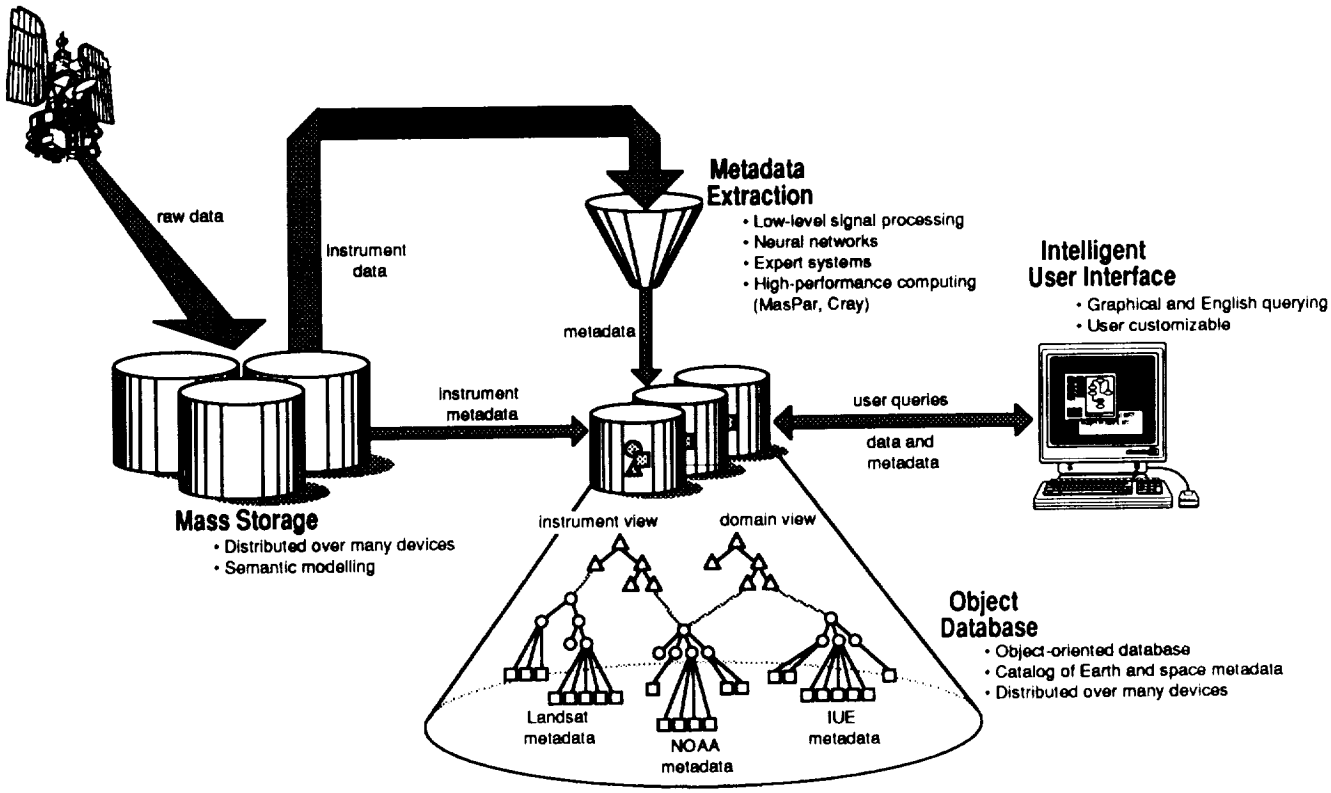


Figure 2: The high-level architecture of the Intelligent Information Fusion System.

1.4 A solution

A design for a spatial data structure suitable for a large, heterogeneous image database with global coverage must account not only for the goals of Section 1.1, but also the difficulties introduced by the richness of the remote sensing domain:

- *Multiple image orientations,* due both to different satellite orbits, and because there is no such thing as "fixed orientation" on the surface of a sphere.
- *Multiple image shapes,* due to the variety of sensors, the tilt of the individual spacecraft, and the alteration of the image border by geometric correction.
- *Multiple image sizes* in terms of the extent of the image boundaries on the surface of the sphere: e.g., sensors mounted on airplanes have smaller fields of view than similar sensors mounted on orbiting platforms.

The data structure described in this paper, together with the supporting expert systems for ingest and querying, addresses all these concerns. The result is a *spatial data handling system* which can handle NASA's next generation of image catalogs.

2 Simplifying spatial queries by a transformation scheme

2.1 The general concept

A variety of spatial data handling problems in complex spatial domains can be solved by mathematically transforming the domain D into a new domain D' where the corresponding queries can be handled more efficiently [Same90, p. 186]. Such transformations map a complex object in D (in this case, an image) into a single point in D': this point is referred to as the object's *representative point.* We are then left with the simpler goal of designing a data structure that can handle the storage and query of points rather than arbitrary shapes. Two difficulties with this approach can be encountered:

- A query region R in D must be transformed into its equivalent R' in D', and R' may be difficult to generate or to calculate with, even for simple R.
- The transformation may result in some loss of information about the stored objects, so that additional computation may be needed to exactly satisfy a spatial query.

These difficulties are dealt with in Section 4, where the implementation of the data structure is described.

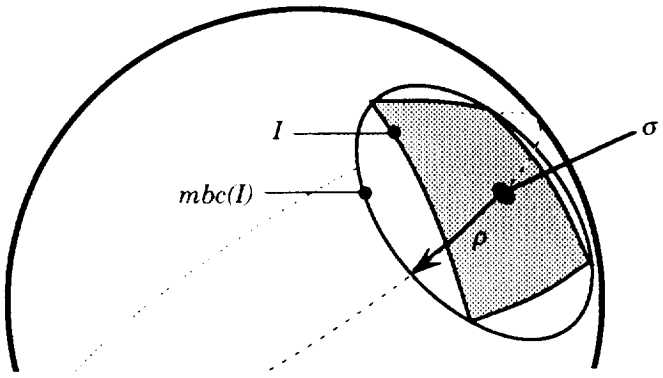


Figure 3: The minimal bounding circle of an image on the globe. Note that the radius is measured along a great-circle arc, like all distances on the surface of a sphere.

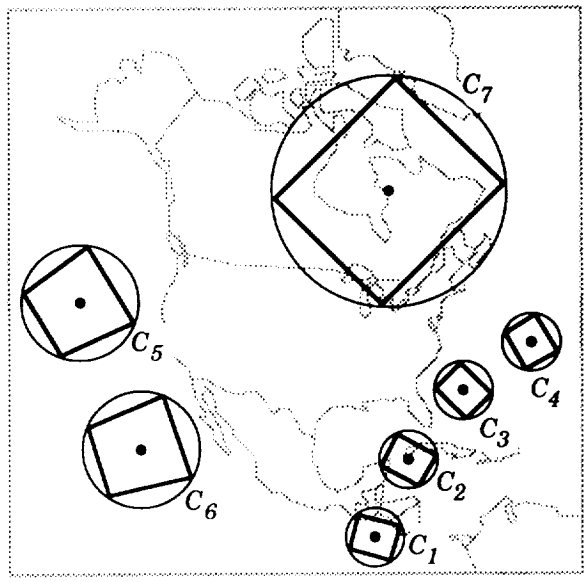


Figure 4: A portion of D, showing a group of images and their minimal bounding circles.

2.2 A transformation scheme for image data

In order to transform images into points, we discard the actual boundaries of the image and concern ourselves only with its minimal bounding circle, which we shall call the *representative circle* for the image (Figure 3). This is closely related to the approach taken by [Oost90], which takes the minimal bounding circles of objects on a planar surface instead of on a spherical surface. Note that the "representative circle" approach eliminates the problems of multiple image shapes and orientations.

By treating images as circles, we are able to describe every image by only two parameters: the location of the circle's center, which we shall denote as σ, and the radius of the circle, which we shall denote as ρ. Thus, every image can be treated as a simple point $\langle \sigma, \rho \rangle$. Under the terminology of [Hinr83], σ is the point's *location* parameter, and ρ is the point's *extension* parameter.

2.3 Visualizing the transformation

Consider a part of the globe over which several images have been taken, shown in Figure 4. For illustration purposes we will show only a small part of the globe so that it may be rendered as a simple plane, although it must be stressed that what is actually being shown is a portion of a curved surface. This would represent a scenario in D.

To map this scenario to D', we compute for each image I_i the center σ_i and radius ρ_i of its minimal bounding circle, and plot the resulting point $\langle \sigma_i, \rho_i \rangle$ in D' as shown in Figure 5.

To more compactly illustrate what the space of D' looks like, we must make some diagrammatic simplifications. Figure 6 shows how the surface of a sphere can be mapped onto the perimeter of a circle by means of a *space-filling curve*. This is a single curve that begins in the diagram at point A, passes through every point B, C, D, etc. on the sphere, and eventually returns to A (also labeled Z in the diagram). The curve places an ordering on the points: A is before B, B is before C, etc., and this ordering enables us to place every point on the sphere's surface onto the perimeter of the circle below. Note that points which are close to each other on the circle (like B and C) correspond to points which are close to each other on the sphere.

By using this mapping, the scenario of Figure 5 is depicted again in Figure 7. Here, D' is shown as the surface of a cylinder: the position on the vertical axis represents the ρ value, and the position along the circular perimeter represents the σ value.

Since individual sensors can be expected to produce large numbers of images of the same size, we expect the distribution of representative points for a large, heterogeneous image database not to be uniform, but instead to be concentrated in different strata along the ρ axis (Figure 8).

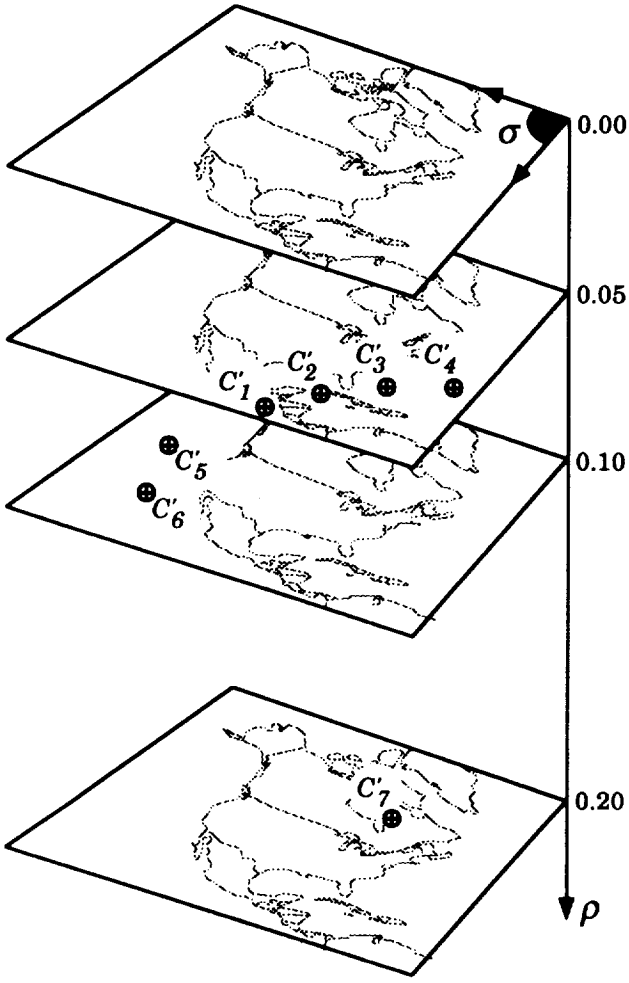


Figure 5: The representative points of the images in Figure 4, plotted in a portion of D'.

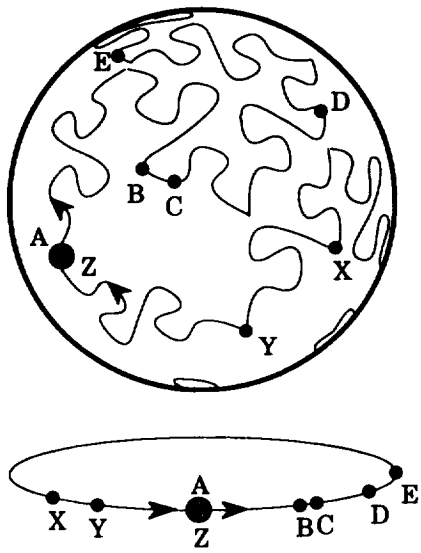


Figure 6: How the surface of a sphere (above) can be mapped onto the perimeter of a circle (below) by using a space-filling curve $A, B, \ldots, Y, Z$. For clarity, the curve on the sphere is not shown in its entirety.

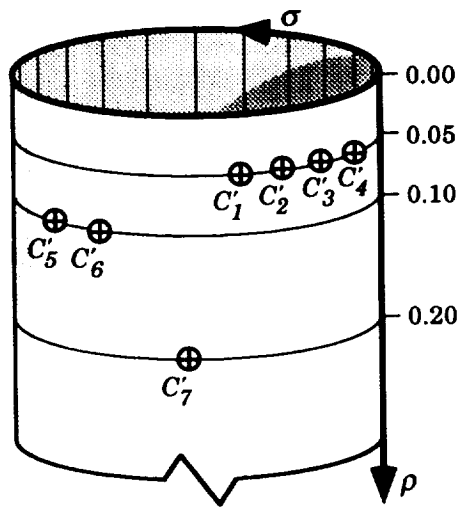


Figure 7: The same representative points as in Figure 5, this time plotted on a cylinder to represent D' more compactly. Every circular cross-section of this cylinder represents the entire surface of a sphere (the globe).

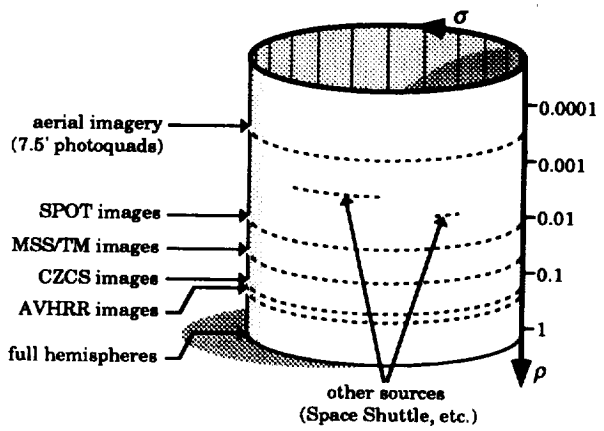


Figure 8: The expected distribution of representative points in D'. For convenience, ρ is shown on a logarithmic scale.

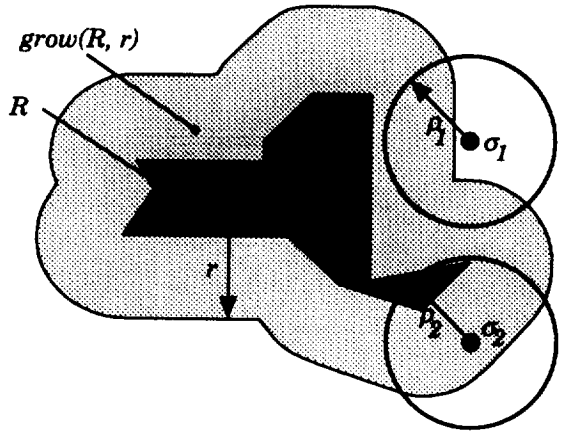


Figure 9: Criteria for a representative circle to overlap R. Both circles have the same radius, $\rho_1 = \rho_2 = r$, but different locations.

3 Processing queries in the transformed space

3.1 Processing window queries

Given a query region R on a sphere, we note that the further the center of a circle C is from R, the larger the radius of C must be if C is to overlap R. Let $grow(R, r)$ denote the locus of all points that are within a distance of r from R: read this as "grow R by radius r" A sample R and $grow(R, r)$ are depicted in Figure 9. We observe:

> A representative circle $C_i = \langle \sigma_i, \rho_i \rangle$ overlaps a region R if and only if its center σ_i falls inside $grow(R, \rho_i)$.

This rule is demonstrated in Figure 10, which depicts in D a query region R, the representative circles for four images $C_1 \ldots C_4$, and the region $grow(R, \rho_i)$ for the various image radii ρ_i. Note that:

- σ_1 is not inside $grow(R, \rho_1)$, and σ_3 is not inside $grow(R, \rho_3)$. Therefore, neither C_1 nor C_3 overlap R.
- σ_2 is inside $grow(R, \rho_2)$, and σ_4 is inside $grow(R, \rho_4)$. Therefore, both C_2 and C_4 overlap R.

Now, consider Figure 11. It depicts in D' the representative points $C_i' = \langle \sigma_i, \rho_i \rangle$ for the images in Figure 10. For each point the corresponding region $grow(R, \rho_i)$ has been plotted, on the same cross-section of D' where C_i' resides. Notice that, if $grow(R, r)$ had similarly been plotted for all r in ρ, the regions would trace out a cone-like solid in D'. In terms of formulating window queries in D', this means that:

> An image's circle in D overlaps a region R if and only if its representative point in D' falls within the cone in D' whose cross-section at $\rho = r$ is $grow(R, r)$.

If the region R is a single point p, then this becomes the definition for a point query, where $grow(p, r)$ is simply a circle with center s and radius r.

[Same90, pp. 187-192] observes that, when employing transformation schemes which represent stored objects by points that have distinct location and extension parameters, window queries and containment queries generally produce cone-like search regions. This is also true of the model described above (Figure 12), and for this reason we refer to the search regions in D' as *search cones.*

3.2 Processing containment queries

Up to this point we have dealt with window queries, which produce cone-like search spaces in D'. A containment query's search space is also a cone-like region, but differing in the way a cross-section of the cone is defined for a given value r of ρ: instead of its being the locus of all points p such that *any* point of R is within a radius of r from p, it is the locus of all points p such that *all* points of R are within a radius of r from p. Call this cross-section $cover(R, r)$.

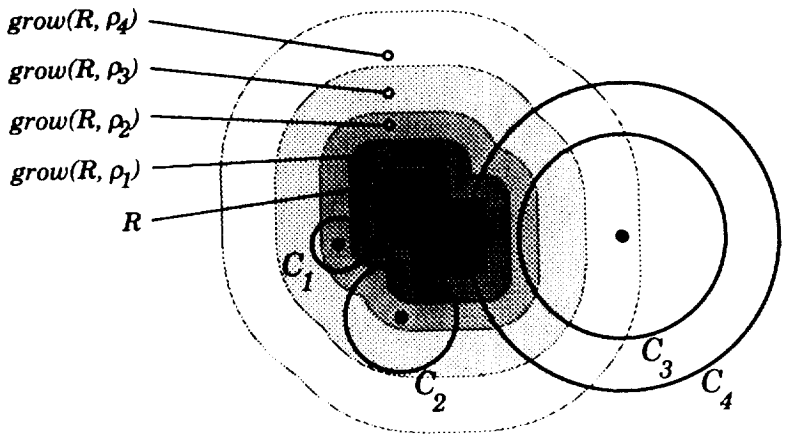


Figure 10: Four representative circles, as they would appear in D. Also shown are $grow(R, \rho_i)$ for each circle C_i.

Whereas $grow(R, r)$ is relatively easy to compute even on a sphere, $cover(R, r)$ is more complex. But, as it turns out, we need never compute $cover(R, r)$ directly to process queries.

To begin, notice that since an image with radius $\pi/2$ is a full hemisphere, we need not concern ourselves with images where $r > \pi/2$. It can be shown that for all $r \leq \pi/2$, if all the *vertices* of R are within a radius of r from a given point p, then the edges between those vertices are completely within a radius of r from point p, and thus *all* of R is within a radius of R from point p. So $cover(R, r)$ is actually the locus of all points within a radius of r from every vertex of R. Therefore, if R has n vertices, $cover(R, r)$ is the intersection of n circles of radius r whose centers are at the vertices of R (Figure 13).

Let R have vertices $v_1 \ldots v_n$. In terms of containment queries, this means that:

> An image's circle in D completely contains a region R if and only if its representative point in D' falls inside *all* the search cones $S_1 \ldots S_n$, where the cross-section of S_i at $\rho = r$ is $grow(v_i, r)$.

The search cone for the containment query can thus be defined as the intersection of n search cones: the search cones for the point queries on the n vertices of R (Figure 12).

4 The hypercylinder data structure

4.1 Design issues arising from implementation details

At the core of the spatial data handling system is the data structure that stores and retrieves points in D', named the *hypercylinder* because of the shape of the transformed space. To ensure that it is capable of efficiently processing queries in D, we must consider factors that place practical limitations on how the corresponding search regions in D' can be manipulated.

Although the cross-section of a search cone at a given value of ρ is easy to generate, computations involving the cone itself require a great deal more processing. Therefore we shall handle queries by dividing the search cones into cross-sections that may be dealt with individually (Figure 11 provides an illustration of "slices" of a search cone). The ramification for the data structure is that D' must be represented internally as a collection of slices that can be queried independently.

Since we must still compute cross-sections for each slice of the data structure, we need to divide D' into a manageable number of slices. If slices are infinitely thin (i.e., the data points in a given slice all have the same ρ value), then even small variations in the extents of images will result in a need for a large number of slices. We thus let each slice cover a range of ρ values.

To execute a query (Figure 14), we handle one slice of the search cone at a time, and then merge the results together to form the final result. For each slice, we com-

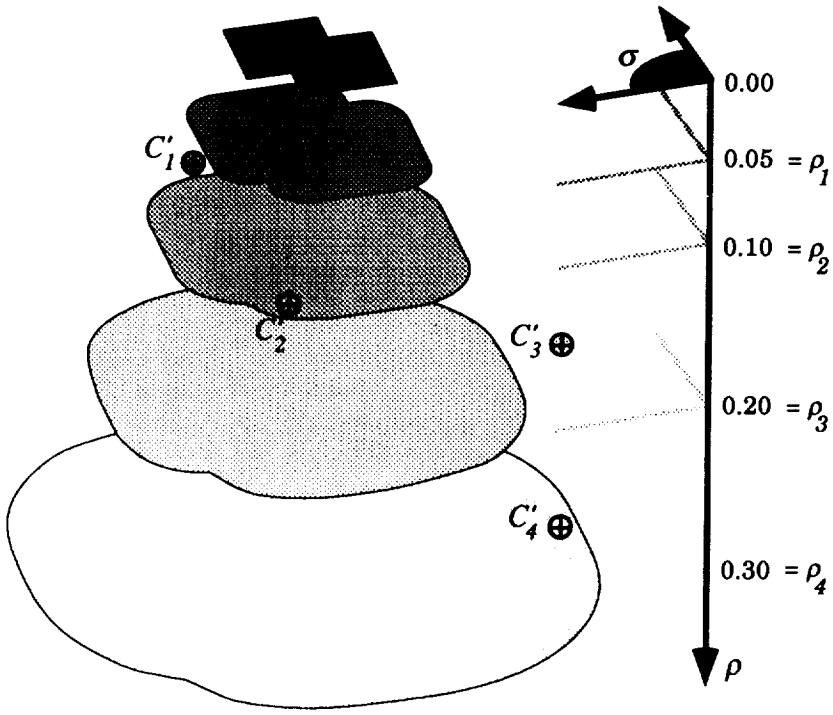


Figure 11: The representative points for the circles in Figure 10, as they would appear in D'. Also shown are $grow(R, \rho_i)$ for each circle C_i'.

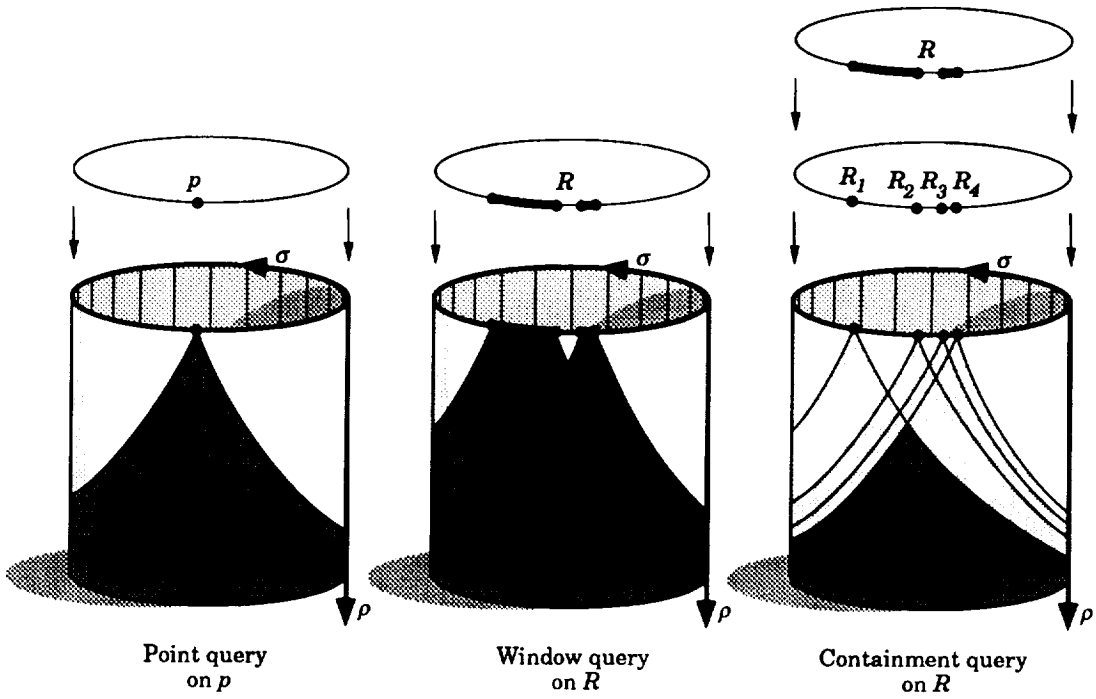


Figure 12: The search cones (shaded) in D' for a point query, window query, and containment query. Notice that the search cone for the containment query is the intersection of four search cones for point queries.

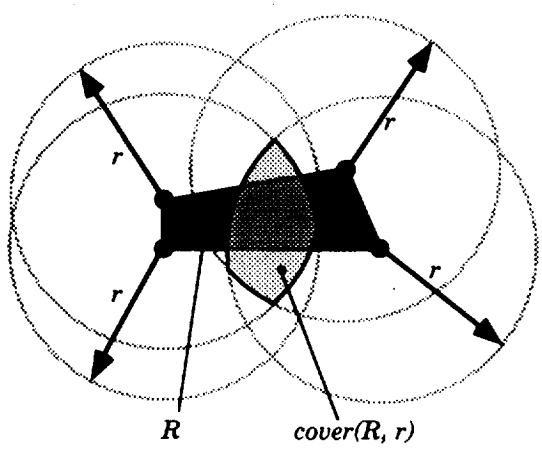


Figure 13: A region R and $cover(R, r)$. Any circle of radius r in $cover(R, r)$ will overlap all points in R.

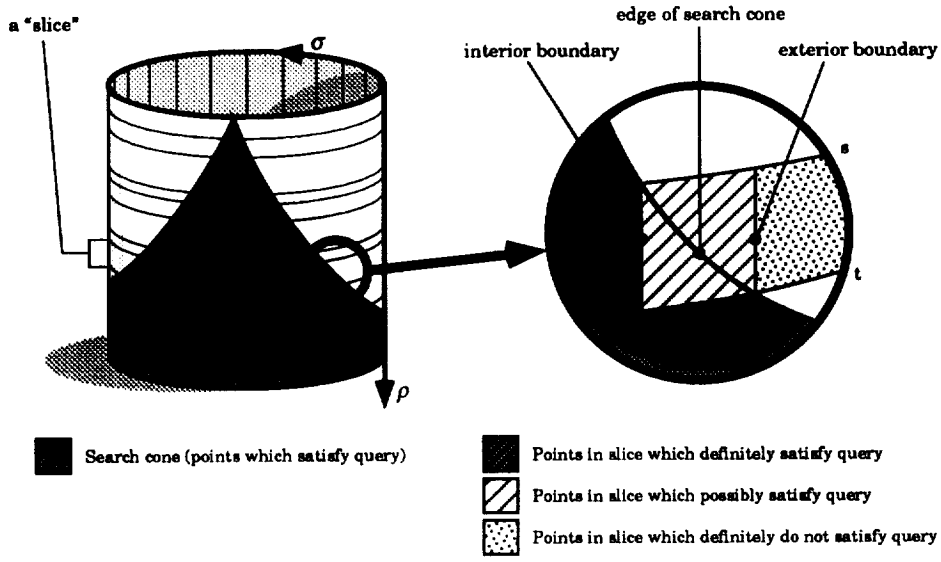


Figure 14: Slicing up D', and approximating the portion of a search cone inside the slice from $\rho = s$ to $\rho = t$. By computing the interior and exterior boundaries of the search cone in that slice, we can divide the points in that slice into three groups – those that definitely satisfy, possibly satisfy, and definitely do not satisfy the query.

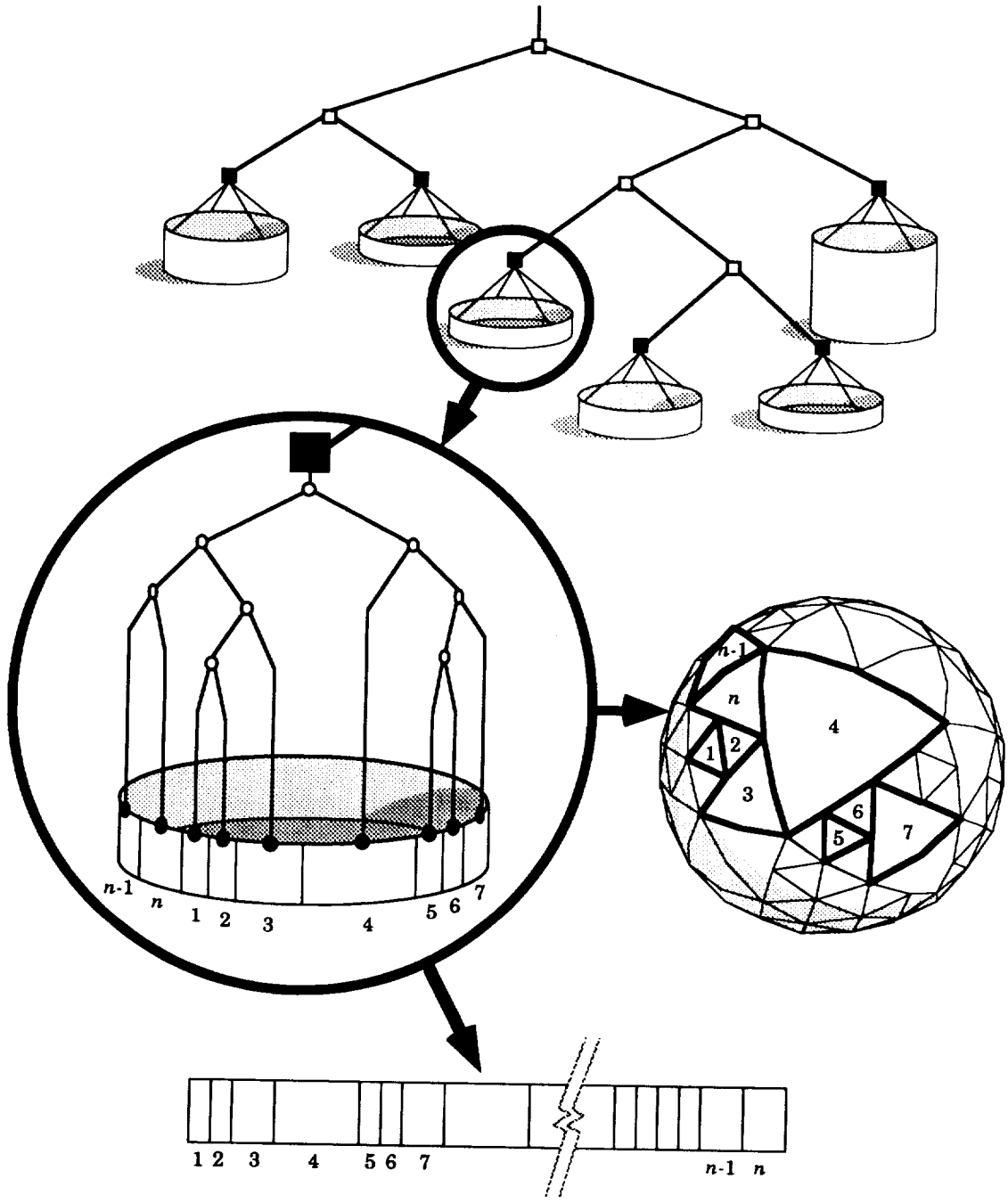


Figure 15: The hypercylinder data structure. Square nodes belong to the BST for ρ, circular nodes belong to the SQTs for σ, white nodes are internal nodes, and black nodes are leaf nodes. A close-up of one slice is depicted, with n leaf nodes in its SQT. Also shown is how some of the SQT leaf nodes (numbered) might look if the surface of the slice were "unrolled" (bottom), and how the corresponding trixels might look on the surface of a sphere (right).

pute *two* cross-sections of the query's search cone: one where the cone passes through the top of the slice, and one where it passes through the bottom of the slice. These two cross-sections give us, respectively, the interior and exterior boundaries of the search cone as it passes through the slice. We observe that:

- Data points from inside the interior boundary are *definitely* inside the search cone.
- Data points from between the interior and exterior boundaries are *possibly* inside the search cone, and must be tested on an individual basis.
- Data points outside the exterior boundary are definitely outside the search cone.

Notice that the thicker the slice, the greater the difference between the interior and exterior boundaries, and hence the more data points we can expect to have to test in this region – which we shall call the "possibly-satisfy" region – during a query. To maximize query efficiency, we must slice up the hypercylinder so that areas of D' with many data points are sliced thin, while areas of D' with very few data points may be covered by thick slices since fewer points will need to be tested in those areas. As revealed in Figure 8, we expect the representative points for images to be largely concentrated in different "strata" of D'. Unfortunately we cannot predict where all such strata will eventually lie, due to the continuous launching of new sources of image data.

As the number of points within a slice grows, eventually the density of points within that slice is such that excessive time is spent deciding whether to accept a point within the "possibly-satisfy" region. At this time, the slice must be split so that the collection of points within the subslices is more homogeneous. A heuristic approach to recognizing when this division should occur and where the division should be made is given in Section 5.1.

4.2 The data structure design

For an overview of the hypercylinder's design, refer to Figure 15. The top-level view is a binary search tree (BST), whose branches discriminate between values of ρ and whose leaves are the slices of D'. The data structure at each leaf is a sphere quadtree (SQT) [Feke84] [Feke90], a special variation of a quadtree designed for storing and retrieving points distributed on the surface of a sphere. The branches of a SQT discriminate between values of σ and the leaves represent triangular regions of the globe (Figure 16). The representative points of images are stored in the leaves of each SQT.

The sphere quadtree is a unique data structure in that it models the globe without introducing distortions

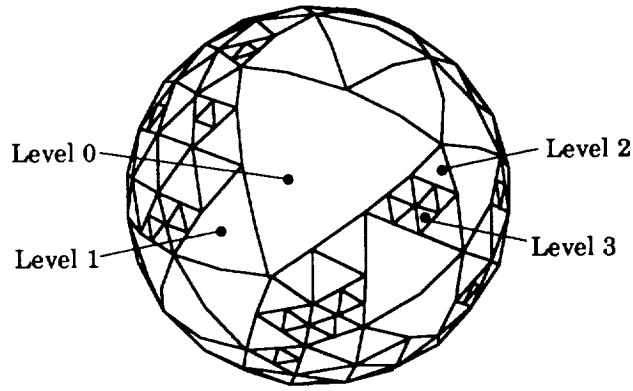


Figure 16: How a sphere quadtree divides the globe into triangular patches (called *trixels*). The higher the level number of a trixel, the deeper in the tree it is, and the smaller the area it covers.

or discontinuities, as other approaches such at latitude-longitude based schemes do (see Section 1.2). Conceptually, it divides the sphere into twenty identical equilateral triangles called *trixels*, where each trixel is a "bucket" for data points. When a trixel reaches its threshold number of data points, it is split into four nearly equal-area subtrixels. This subdivision is called *refinement* since it produces smaller trixels which, like the pixels in an image, can represent regions to a higher degree of resolution. As with most spatial data structures, refinement in a SQT can continue indefinitely: the result is that areas of the globe that are densely populated are more refined, so query regions in those areas are more accurately represented by the higher resolution trixels in the SQT.

Since satellite orbits generally provide global coverage, we expect the SQT for each slice to be fairly equally refined over most of the surface of the globe, i.e., the SQT is well balanced, and so spatial queries are handled with similar efficiency regardless of their location. But since satellite orbital paths are often designed to produce fully-overlapping images at each pass over a location, a clustering of the representative points occurs and results in a tree that is *globally* well-balanced, but *locally* unbalanced. A means for overcoming this problem exists, and is discussed in Section 7.1.

To ensure that the slices are split in an optimal manner when they achieve their threshold number of data points, a *profile* of how the data points are distributed in each slice is maintained. These profiles are used as heuristic devices to determine where the slices should be

subdivided. Their actual implementation is described in Section 5.1.

5 A spatial data management system

The primary motivation for designing an entire spatial data management *system* for the remote sensing domain, as opposed to just the custom-tailored spatial data structure described above, is that domain knowledge can often be employed to improve the overall performance of *any* data management scheme. For a remote-sensing catalog, such knowledge encompasses:

- *Model information:* what real-world entities and concepts (observations, sensors, geographic regions, scientific parameters, classification schema) are represented in the catalog, and what sorts of questions may be asked about them by end-users. This is mostly declarative information, intended for use by both the end-users and the system. It allows the users to ask the system about its contents, and it enables the system to translate users' natural-language and graphical queries into the system's internal representation.

- *Data structure information:* what data structures (e.g., the hypercylinder) exist in the database, under what conditions they should be used (e.g., spatial queries), and how data is distributed in them (e.g., the profiles mentioned in Section 4.2). This procedural and declarative metaknowledge is used by the catalog to construct plans for queries, to generate the necessary calls to the catalog's underlying database management system, and to optimize the query plans as intermediate results are returned.

- *Operational information:* the performance of the hardware devices over which the database is distributed, the anticipated system loads over the course of a typical day or week, the types of queries most frequently made, etc. This is largely declarative information, used by the catalog in performing a variety of tasks ranging from query optimization to automatic data structure reorganization.

The spatial data management system makes use of such information in its two supporting expert systems: the Spatial Ingest Expert System and the Spatial Query Expert System, both of which are discussed below.

5.1 The Spatial Ingest Expert System (SIES)

The primary function of the SIES is to govern the splitting of the slices of the hypercylinder, ensuring that each slice is divided so that dense strata of D' end up in thin slices, with thick slices covering the sparser expanses of D'. As mentioned in Section 4.2, each slice maintains a profile of the *current* distribution of the data points in the slice. In addition to this, the SIES incorporates information about the expected *future* distribution of points. Both provide a heuristic means of optimizing the hypercylinder as it is being built.

The primary requirements for profiles are that they must be easy to update during ingest, be implemented to allow rapid calculation during splitting, be large enough to adequately capture the distribution of points in a slice, and be small enough not to incur a large storage overhead. We have experimented with an approach based on incremental sampling of the representative points as they are stored in the slice: the profile consists of a small reservoir of the ρ values of sampled points, and as each new point is ingested there is a chance that one of the current elements of the reservoir will be replaced by the ρ value of the new point. During splitting, the profile is analyzed to determine where the ρ values are clustered, indicating emerging strata in D'.

The profiles are supplemented in the knowledge base by the *bias list*: a list of strata into which D' is expected to be organized. The bias list is updated whenever a new instrument is added to the knowledge base, and contains the expected minimum and maximum radii of images that the instrument will generate plus an estimate of how many images will be generated over the lifetime of the instrument. Although the bias list can supply information on where a slice is best split (or even whether to defer splitting a slice), its contents do not reflect the actual state of the data structure, and thus neither it nor the profiles are expected to provide maximal performance in isolation.

The hypercylinder initially consists of a single slice covering all of ρ. When the number of points stored in any slice reaches the threshold value for splitting, a strategy for dividing the slice is formulated from one of several alternatives, such as:

- Place the largest cluster into its own slice, and the spaces to either side of this cluster into two additional slices (Figure 17).

- Place the largest expanse of sparsely-populated space into its own slice, and the spaces to either side of it into two additional slices.

- Given an entry in the bias list whose minimum and maximum radii fall within the slice and whose esti-

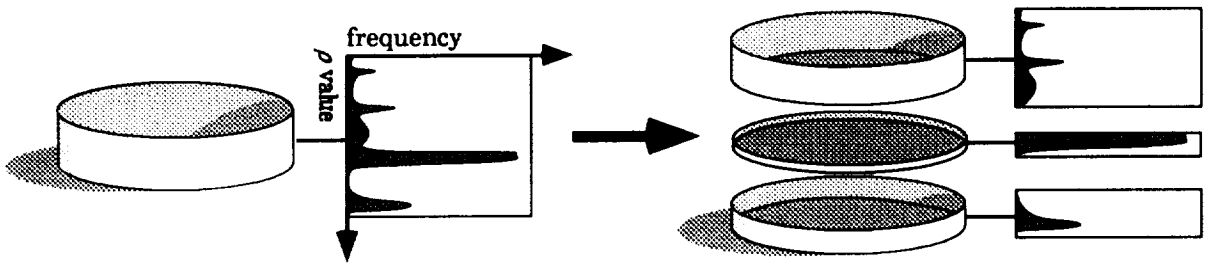


Figure 17: One strategy for splitting a slice, based on the distribution of points in the slice: this "fences in" the largest cluster of points, putting it into its own slice.

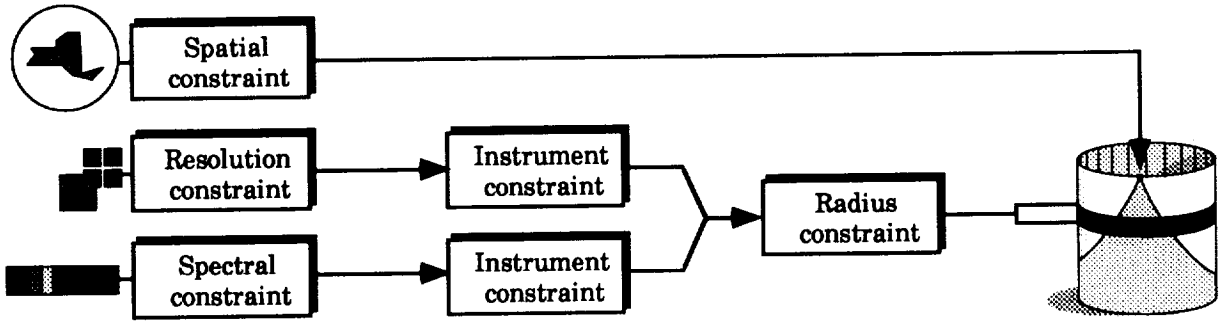


Figure 18: How non-spatial components of a query can place implicit spatial constraints. Only the shaded layers of the hypercylinder must be searched.

mated number of images is high, place that range of values into its own slice and the spaces to either side of it into two additional slices.

- Split the slice so that equal numbers of points are in each subslice.

The strategy chosen depends on factors such as how definite the clusters are, how widely they are distributed, and whether one cluster is significantly larger than the others. Each fact lends weight to one or more of the strategies during selection: these weights are then adjusted as more is learned about the performance of the SIES under the real-world environment.

5.2 The Spatial Query Expert System (SQES)

The SQES is actually a conceptual subset of the larger Query Processing Expert System, a mostly-procedural-knowledge base whose content is the model and data structure information described in Section 5, and whose purpose is the translation, optimization, and execution of user queries. The SQES handles those parts of the task that relate to spatial searches.

The first place the SQES is invoked is during the parsing of queries with symbolic spatial components. In natural-language and menu-driven queries, such components might appear as the names of geographic, political, or climatological regions on the Earth (or as the names of stellar objects or constellations, depending on the catalog type). The SQES translates these terms into geometric region descriptions, possibly invoking external information sources in the process, such as databases that house geopolitical boundaries, or that store the names of astronomical entities under different labelling schemes to allow translation from one scheme to another (e.g. the SIMBAD database).

Figure 18 shows how the SQES can optimize the spatial search by inferring additional spatial constraints from the user's query. The user's specification of desired ranges of image resolution and spectral bands constrains the set of instruments that might be sources of the desired data, which in turn limits the possible sizes of images that can be returned from the user's query, which in turn pinpoints the only slices of the hypercylinder that need to be searched.

The SQES also assists in planning complex spatial queries, where the order in which subparts of the query are executed can play a dramatic role in decreasing processing time. Consider the processing of a containment query: as noted above, containment queries are best handled as the intersection of a collection of point queries. Throughout the system, computing the intersection of a group of unknown sets is performed in a strategic manner: the members of the group are retrieved sequentially, from smallest to largest estimated size, and the most recently retrieved set is intersected with a running "result" set. The system stops and returns the empty set as the result of the intersection if any of the retrieved sets is the empty set.

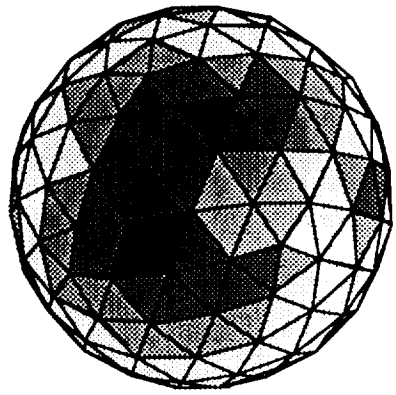


Figure 19: The profile used by the SQES. Darker trixels indicate that observations are more dense in those portions of the globe.

To allow the SQES to estimate the relative sizes of sets returned by the components of a spatial query, yet another profile is kept in the knowledge base. Whereas the previously-discussed profiles represent the distributions of the image *radii*, this new profile represents the distribution of the image *centers* on the globe, giving in effect the "density" of observations around the surface of the globe. Since it associates spherical locations with density values, this profile is implemented as a small spherical quadtree (Figure 19). When the SQES is confronted with a set of query regions that must be ordered by expected content, the area of each region is computed and multiplied by its average density from the profile to produce an estimate of the number of data points in the entire region, and it is by this estimate that processing order is determined. We intend to install similar profile-based approaches for aiding the construction of query plans on top of *all* the catalog's principle data structures.

6 Results

The spatial data handling system has been tested using a portion of the metadata stored in the Pilot Land Data

System (PLDS) catalog. Approximately 3,000 records of TM, MSS and AVHRR metadata were ingested from a flat file into the hypercylinder's SIES via a C program which extracted the appropriate fields for each image's location, boundaries, and primary key. To assess the performance and extendability of the SIES under different implementation languages, it was written both in Quintus Prolog and in CLIPS, an expert system shell developed by NASA's Johnson Space Center and capable of being linked into a C program and accessed via simple function calls. The SIES sends the appropriate ingest requests to the hypercylinder "server": a C++ program containing the hypercylinder data structure, accessible through a TCP/IP socket on a Sun-4. All of the spatial search routines, as well as the profiles for the hypercylinder's slices, were implemented in C++ and reside in the server. Query requests are sent to the SQES, a CLIPS/C module that uses a small, array-based version of the SQT (called a *linear SQT*) to store the SQES profile. The SQES performs the necessary query planning and sends the various partial spatial query requests to the hypercylinder server, which keeps track of execution times for various tasks.

The spherical quadtree components of the system and the supporting spherical geometry routines have been implemented and tested independently inside the IIFS catalog's database. The catalog uses the Smalltalk-based GemStone DBMS, a commercial object-oriented DBMS available from Servio Logic Corporation, which is capable of invoking external C and C++ functions.

The rationale for initially implementing the full hypercylinder as an in-core data structure rather than inside the catalog database was twofold. First, it enabled us to seamlessly integrate the hypercylinder with the C++ spherical geometry objects (such as query regions) and routines (such as *grow()*) necessary for spatial query handling. We found C++ to be an excellent programming platform for rapid data structure prototyping, and are currently using Oregon C++ from Oregon Software, which conforms to the base ANSI documents for this language and thus should produce highly portable code. Second, initial implementation and testing in core enabled us to take CPU-time measurements without concerning ourselves with the I/O and CPU overhead that would be introduced by interfacing with a DBMS.

The *grow()* routine performed well for any given radius: the algorithm is $O(n)$, where the inputs are the n vertices of the query region R and a radius of expansion r, and the outputs are the m vertices of $grow(R, r)$, where $n < m < cn$ for a predefined constant c. Unfortunately, the grown query region is almost always self-overlapping, and some necessary computations (such as determining whether a point is inside $grow(R, r)$) take $O(n^2)$ to process using our current algorithms. Removing the self-intersections from a spherical region appears to be an $O(n^2)$ operation in the best case: we are therefore focusing our attentions on developing more efficient algorithms for manipulating the self-overlapping regions.

7 Future research

7.1 The hypercylinder

The hypercylinder data structure, designed to meet stringent ingest and query requirements for large image catalogs, is nevertheless only one possible data structure and is specifically designed for image data. We hope in the near future to:

- Produce additional spatial data structures customized for efficient storage and retrieval of other types of observations, such as observations in atmospheric domains with additional spatial search criteria such as "altitude."
- Implement these data structures fully inside the catalog's database, ensuring that the hypercylinder's components are clustered so as to minimize page faults during tree traversal.
- Introduce tree compression techniques for the SQTs, as per [Ohsa83], to eliminate the clustering problem mentioned in Section 4.2.

7.2 The Spatial Ingest Expert System

In future implementations, we plan to expand the role of the SIES in the spatial data ingest process. The SIES will be empowered to:

- Periodically survey the data structure for conditions that would compromise efficiency, such as tree imbalance. If such conditions are detected, the SIES must determine how best to reorganize the data structure, and notify the database administrator (DBA) of the problem.
- Estimate the amount of system resources that a reoptimizing step will take, and, based on profiles of system loads, suggest to the DBA the best times for self-correction.
- Maintain a history of major decisions affecting the data structure: when a slice was split and why, when the data structure had to be reoptimized and why, etc. Alert the DBA if it is determined that some subset of the rules has contributed to poor decisions.

7.3 The Spatial Query Expert System

Much of the spatial query optimization is intended to be handled by the catalog's proposed Query Planning and Execution Module (QPEM), in which the SQES knowledge will reside. However, there are still spatial search strategies unique to the SQES that have yet to be explored:

- Transfer more spatial query processing control from the hypercylinder to the SQES. This would involve maintaining a collection of density profiles, each covering a different slice of the hypercylinder. Spatial queries would be handled and optimized independently by each slice of the hypercylinder, based on local profile information.
- Allow the user to specify different levels of spatial query processing. Since many stages of query processing in the data structure divide the tree into three types of branches – definitely satisfies query, possibly satisfies query, and definitely does not satisfy query – the user can be given the power to trade precision for execution time by deciding to either accept, reject, or vigorously test the "possibly satisfies query" branch.

8 Summary and conclusions

The research presented in this paper is intended to serve as the foundation for a new generation of spatial data management systems at NASA, tailored for the general remote-sensing domain and robust enough to support efficient spatial searches regardless of the shape or location of the user's area of interest.

The hypercylinder's two controlling expert systems, the SIES and the SQES, are as necessary as they are novel. By using rule firings in a supervisory expert system to activate data management tasks, the conditions under which different data management strategies are employed can be easily monitored, evaluated, and altered to fine-tune system performance. This is a major step beyond conventional catalog schemes, where inspection and evaluation of the underlying data structures are at best extremely difficult, and adjustment of the associated algorithms is traditionally impossible without down-time for code recompilation.

9 Acknowledgements

The authors would first like to thank William J. Campbell for his continual support of IDM, and for the foresight he has shared with us, which has helped make the IIFS into what we believe to be the finest prototype in existence for NASA's future data management systems. In addition, the research presented in this paper would not have been possible without the inspiration provided by three other researchers: Dr. George Fekete, who developed the spherical quadtree data structure, Nick Short, Jr., whose introduction of planning-and-scheduling technology to the IIFS led to the concept of the SQES, and finally Dr. Samir Chettri, whose expertise in statistics has contributed to the algorithms underlying the SIES.

References

[Camp89] W. J. Campbell, S. E. Hill and R. F. Cromp, "Automatic labeling and characterization of objects using artificial neural networks," *Telematics and Informatics,* Vol. 6, Nos. 3/4, pp. 259-271, 1989.

[Camp91] W. J. Campbell, N. M. Short, Jr., L. H. Roelofs and E. Dorfman, "Using semantic data modelling techniques to organize an object-oriented database for extending the mass storage model," *42nd Congress of the International Astronautical Federation*, Montreal, PQ, Canada, 1991.

[Cromp91] R. F. Cromp, "Automated extraction of metadata from remotely sensed satellite imagery," *Technical papers, 1991 ACSM-ASPRS Annual Convention Proceedings*, Vol. 3, pp. 111-120, 1991.

[Dorf91] E. Dorfman, "Architecture of a large object-oriented database for remotely sensed data," *Technical papers, 1991 ACSM-ASPRS Annual Convention Proceedings*, Vol. 3, pp. 129-143, 1991.

[Feke84] G. Fekete and L. S. Davis, "Property spheres: a new representation for 3-d object recognition," *Proceedings of the Workshop on Computer Vision: Representation and Control*, Annapolis, MD, 1984.

[Feke90] G. Fekete, "Rendering and Managing Spherical Data with Sphere Quadtrees," *Proceedings of the First IEEE Conference on Visualization,* IEEE Computer Society Press, Los Alamitos, CA, 1990.

[Hinr83] K. Hinrichs and J. Nievergelt, "The grid file: a data structure designed to support proximity queries on spatial objects," *Proceedings of the WG '83 (International Workshop*

on Graphtheoretic Concepts in Computer Science), M. Nagl and J. Perl, eds., Trauner Verlag, Linz, Austria, 1983.

[Ohsa83] Y. Ohsawa and M. Sakauchi, "The BD-tree – a new n-dimensional data structure with highly efficient dynamic characteristics," *Information Processing 83*, R. E. A. Mason, ed., North-Holland, Amsterdam, 1983.

[Oost90] P. van Oosterom and E. Classen, "Orientation insensitive indexing methods for geometric objects," *Proceedings of the 4th Intenational Symposium on Spatial Data Handling*, Vol. 2, Zurich, Switzerland, 1990.

[Same90] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, New York, NY, 1990.

[Short91] N. Short, Jr., "A real-time expert system and neural network for the classification of remotely sensed data," *Technical Papers 1991 ACSM-ASPRS Annual Convention*, Vol. 3, pp. 406-418, 1991.

Data Exploration Systems for Databases

Richard J. Greene and Christopher Hield
Environmental Assessment and Information Sciences Division
Argonne National Laboratory
9700 South Cass Avenue * EID/900
Argonne, Illinois 60439-4832

ABSTRACT

Data exploration systems apply machine learning techniques, multivariate statistical methods, information theory, and database theory to databases to identify significant relationships among the data and summarize information. The result of applying data exploration systems should be a better understanding of the structure of the data and a perspective of the data enabling an analyst to form hypotheses for interpreting the data. This paper argues that data exploration systems need a minimum amount of domain knowledge to guide both the statistical strategy and the interpretation of the resulting patterns discovered by these systems.

1. INTRODUCTION

Data exploration systems apply machine learning techniques, multivariate statistical methods, information theory, and database theory to databases to identify significant relationships among the data and summarize information. The result of applying data exploration systems should be a better understanding of the structure of the data and a perspective of the data enabling an analyst to form hypotheses for interpreting the data. In a sense, data exploration systems are a tool of the "scientific method": raw data is collected, laws describing the principal features of the data are hypothesized and tested, and theories explaining the laws are hypothesized and tested by using the theory to predict new information.

The benefits of data exploration systems can be significant. An analyst deluged with data can greatly reduce the time needed to understand the meaning of the data, and the accuracy of data analysis can be greatly increased. One might think of a data exploration system as a tool for examining large and complex data for meaning that might normally go unnoticed.

The purpose of data exploration systems is to reveal structure (or equivalently "pattern") in data. The basic operations of a data exploration system consist of describing, detecting, and searching for structures. Ideally, the detected structure did not arise by chance and can be described in the terms of the subject matter that produced the original data. The statements above naturally imply issues regarding the role of domain specific (i.e., subject matter) knowledge in data exploration.

Collecting data is often much easier than transforming the data into useful knowledge. The result is a lag between the time the data becomes available for analysis and the time knowledge can emerge from an analysis of the data. This problem is aggravated in areas where data is collected about phenomena to discover what factors affect performance and the relationships among the factors. The goals of analysis are not only to improve future performance but also to use the data to understand the underlying principles governing the observed behavior. The time and effort required to analyze a large amount of data representing a variety of qualitative and quantitative attributes is often quite prohibitive. It is from within this context that data exploration systems have emerged.

A previous NASA application of data exploration in 1983 used the AUTOCLASS system for the analysis of the Infrared Astronomical Satellite (IRAS) Data. For one year, the system sampled 94

spectral intensities and 2 celestial coordinates. In all, there were 5,425 records of data. Human data analysts spent two years analyzing and classifying the data into a known, but inadequate, taxonomy. In 1987 the AUTOCLASS program, a domain independent program based on Bayesian statistics, was applied to the IRAS data [Denning]. The program ran for 36 hours and created a new classification scheme and detected statistical patterns in the data that humans interpreted as "discoveries". The overall response to the AUTOCLASS system, however, has been reserved.

This paper presents our research into data exploration systems. The research is oriented toward analyzing databases of historical performance data for patterns indicative of success and failure. The findings presented here are applicable to any data exploration system. The first area discussed is capabilities: which patterns are sought in the data, which techniques identify the patterns, and how the data patterns are presented to a human investigator. Next, a methodology is discussed for effectively applying a data exploration system (i.e., how do the system's characteristics affect how it is applied). The research is a timely contribution to data exploration, as it identifies weak and missing capabilities of these data discovery systems, and, in some cases, the recommendations have been implemented and investigated.

This paper will show that induction and generalization over a database cannot be completely free of domain knowledge. At a minimum, the data exploration system must account for the semantics of numeric data. Next, classical statistical methods must be employed with great care because both the statistical hypothesis and the method of data collection strongly affect the validity of induction and the interpretation of the resulting generalization. Finally, the architecture of a data exploration system reflects the state of knowledge about the problem domain : when little is known about the domain being explored, the system is a loosely coupled set of tools supported by metadata. The more known about the domain, the less exploration there is and the more predictable the analysis becomes.

2. EXAMPLE of DATA EXPLORATION

Consider the following hypothetical database containing data regarding the past performance of a pre-launch rocket fuel monitoring subsystem:

Table 1. Sample Database

Manufacturer	Sensor Type	Launch Time	Number of Sensors	Sensor Indication
ABC Corp.	pressure	morning	6	anomalies
XYZ Inc.	temperature	afternoon	12	clear
ABC Corp.	density	night	6	clear
XYZ Inc.	volume	morning	6	anomalies
XYZ Inc.	volume	morning	12	clear
XYZ Inc.	pressure	afternoon	6	anomalies
ABC Corp.	density	night	12	clear
XYZ Inc.	volume	afternoon	6	clear
ABC Corp.	temperature	morning	6	clear
XYZ Inc.	pressure	night	6	anomalies
ABC Corp.	temperature	morning	12	clear

For simplicity's sake, there are only eleven launch descriptions. Sensors are manufactured by either ABC Corp. or XYZ Inc. The sensors measure one of four possible fuel-related factors: the temperature, pressure, density, or volume. Launch times are categorized as either morning, afternoon or night (after-dark) launches. Sensors are installed in batteries of six, with a maximum of two batteries or twelve sensors. Finally, each data record is "classified" by its reading indication. The sensor indication exhibits two possible readings : clear (i.e., a "successful"

reading) or anomalies (i.e., inconsistent sensor reports). This gives a possibility of (2*4*3*2) 48 different attributes to describe the environment of a sensor reading. The structure of the data is shown in the decision tree in Figure 1. The attributes are shown as nodes and the edges are labeled with the attribute values. The basis for classification is the indications "clear" and "anomalies", shown as plus and minus signs. The ID3 induction algorithm was used to create the tree.

One hypothesis from this analysis is that the sensor type was the most important factor in successful sensor readings, followed by the number of sensors, the manufacturer of the sensors, and finally the external temperature. This observation implies that some launches had only this one factor in common and that this one factor might have contributed significantly to the success or failure of the sensor readings. In this example, the induction algorithm is identifying factors that seem to be responsible for sensor success or failure. Specifically, pressure sensors tended to account for anomalous readings while sensors monitoring the fuel temperature and density tended to read successfully. This observation remains constant regardless of the values of the other attributes. Likewise, sensors monitoring fuel volume with twelve sensors were likely to detect fuel status correctly. However, when monitoring fuel status with six of XYZ Corp's fuel volume sensors, the external temperature became a decisive factor. Morning launches were likely to exhibit anomalous sensor readings while afternoon launches did not. Once the factors are identified, we now must seek verification of the patterns found by generating an explanatory hypothesis (i.e. a hypothesis that explains why the observed pattern is true and thus how to encourage or avoid the pattern in similar situations -- a "lesson learned"). Hypothesis generation is discussed later.

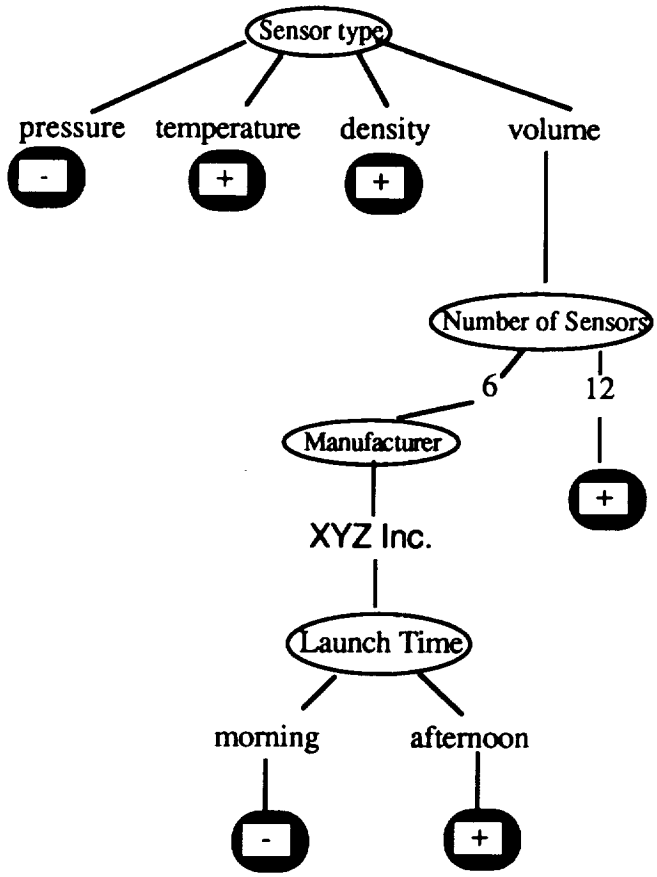


Figure 1. Decision Tree Induced from Table 1.

The structure of the decision tree above can be automatically generated and may serve as a basis for hypothesis generation. For example, why were pressure sensors such a determining factor? One hypothesis is that the fuel pressure varies within the shuttle's rocket boosters between the sensors located at the top of the booster and those at the bottom. This hypothesis could provoke a deeper investigation. It may describe a significant pattern of sensor failure: launches with sensors from different manufacturers, performed at various times of the day, using a different number of sensors, all exhibited anomalous fuel status readings when using fuel pressure sensors. The type of sensor was their only common factor. Likewise, why were morning launches subject to invalid sensor readings? One factor might be the number of sensors. When the number was twelve, readings in similar launches were successful. Thus, a possible explanation of the detected pattern can serve as a hypothesis and lead to important yet subtle new lessons learned which the data supports and yet, perhaps, goes unnoticed or unmentioned by any human analysts. One reason patterns may be hard to detect is the sheer volume of data coupled with the large range of attribute values. Without some form of statistical summary, how can an analyst extract as much information as possible from such high volume data? The automatic inductive analysis described above applies such analyses to the data and creates a discrimination tree as depicted graphically in Figure 1. In the discrimination tree, an analyst can rank the attributes with respect to their ability to classify sensor indications in order to determine which play significant roles in correlating the data as success or failure. In short, the induction algorithm above answers the questions "which attributes were the most significant" and "how are the data related?"

One can see the value of the analysis presented above. Given larger data sets with more attributes and greater attribute ranges, the analysis task would become insurmountable without a tool such as the induction algorithm. It is important to note that the induction algorithm will identify any pattern supported by the data set. However, the pattern may be only coincidental or trivial. If the data have cause/effect, correlations, or other useful information, then the induction algorithm will find it and make the relationships explicit for use in hypothesis generation [Parsaye, Hoaglin].

3. CAPABILITIES of DATA EXPLORATION SYSTEMS

The main issue of data exploration systems is the type of regularities the system can detect. Each system's approach can detect some regularities but is ignorant of other types of regularities. In short, data exploration systems detect regularities that their designers deem important. The domains of applications however, may exhibit certain types of regularity. To what degree can data exploration systems remain domain independent and "generalized"? How should a "regularity" be defined and how can we ensure that the assumptions for its detection are satisfied by the data in question? And, once a pattern has been detected, how should the pattern be interpreted by the user? Hamming's motto for those applying numerical methods also suits those applying data exploration systems : the purpose of computing is insight, not numbers. The choice of computing technique affects how we understand the results [Hamming].

As stated, each system's approach can detect some regularities but is unable to detect others. For example, consider a data exploration system determining the relationship between two variables by applying the chi-square test on a five-by-five contingency table of data values. For the sake of example, assume there are only two real-valued attributes X and Y, and the technique is applied to discover if these two attributes are related and, if possible, describe the relationship. Let the graph of the actual values be the sawtooth wave shown in Figure 2.

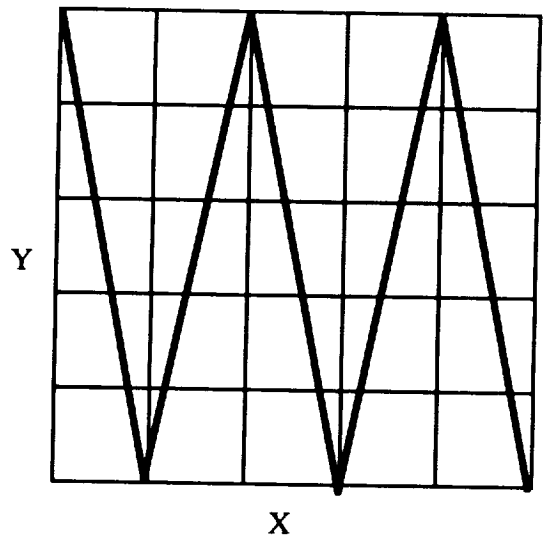


Figure 2. Chi-square test for attribute relationship.

The null hypothesis is that the variables are unrelated; i.e., each bin (or square) is uniformly distributed with data points. To reject the null hypothesis, the observed values must be different from the expected values appropriate to the desired significance level. Yet, as Figure 2 shows, the points are uniformly distributed, as the bins contains approximately the same number of data points. The chi-square test as described would not allow us to reject the null hypothesis. In other words, the attributes are judged to be unrelated because, in this case, the data exploration system fails to detect periodic behavior. If the amplitude or frequency of the wave is varied, the dynamic behavior of the data exploration system reaches critical points where the system can detect a regularity using the chi-square test, but as the amplitude or frequency of the wave crosses critical thresholds such as the one depicted in Figure 2, no relationship is detectable even though the essence of the regularity remains the same. It appears as though the phenomena under examination experience moments of chaos when, in fact, the chaotic behavior is inherent in the detection technique.

As this example suggests, the capabilities of the data exploration system should match the features of the explored context or the user risks "discovering" artifacts of the discovery mechanisms themselves and, thus, inducing invalid generalizations. Yet, matching a technique to problem context can be paradoxical: isn't one of the purposes of data exploration to discover precisely these descriptive features?

Nevertheless, several capabilities can be enumerated. First, the data exploration system should offer a variety of techniques to detect potential regularities and constrain their application appropriately. Without this capability, one cannot be sure that the regularities detectable are the only ones in the data. Multivariate statistical methods offer such a variety and, in addition, constrain when a method applies. Data exploration tools often incorporate many of these sophisticated methods but fail to detect when the techniques do not apply. Thus, the results of applying these techniques can be invalid yet appear as authoritative. The problem of invalidity is discussed in Sections 5 and 6.

Second, the data exploration system should support a flexible and appropriate strategy for data exploration. A *statistical strategy* is a formal justification for the selection, ordering, and application of techniques made during the course of data exploration. The data exploration system should in some sense act as an intelligent practitioner of data analysis [Hand]. Some systems

simply provide a "grab bag" of exploratory techniques that the user selects and applies ad hoc and without a sensible strategy. The worth of hypotheses resulting from such methodology should be suspect. The next section addresses this topic in more detail.

Next, the data exploration system should permit exploration sessions to be "frozen" and resumed. Without this capability, analysis must be conducted in a single sitting. However, an exploration strategy may induce a potentially complex series of probes into the structure of the data and it may not be practical to perform these operations in a single sitting. Furthermore, the system should record the lines of exploration and provide a "replay" capability of the last n operations.

Fourth, the data exploration system should provide a facility for semantic data modeling, as the semantics of the data do affect data analysis. Without this capability, two situations can arise: the relationships and generalizations discovered may be invalid or the system may fail to pursue a semantically rich relationship because two semantically related pieces of data are treated as unrelated by the system. A potentially rich semantic link will be lost. For example, the system can formulate relationships between two numerical variables if these numbers are related in the modeled domain; e.g., pressure and temperature. The importance of data modeling cannot be overemphasized in data exploration and is discussed in more detail in section 6.

Finally, there is the issue of missing or noisy data. How should this data be incorporated into the exploration process? Should there exist a three-valued logic for data values: unknown, not detected, or present? Similarly, some data may not be noisy at all, and other data must be smoothed before analysis. Clearly, a theory of data exploration should address these issues.

4. CONTEXT of DATA EXPLORATION

Next, how does a data exploration system support the analysis process? Some systems such as BACON use empirical numerical data directly to induce a law supplemented with theoretic variables that both simplify the representation of the law and serve as a conceptual aid [Langley]. For example, BACON induced Black's heat law and the concept of "specific heat". Other systems such as IXL operate more interactively by placing a human in-the-loop [Parsaye]. In this case, the data exploration system is more an integrated set of loosely-coupled tools. BACON gives the user an end result while IXL gives the user a set of intermediate results. The style of investigation imposed by the data exploration system should be appropriate to the problem under investigation.

At best, the exploration tool should guide the user in selecting an exploration strategy and then ensure that the data satisfies the assumptions underlying the selected mathematical techniques. Given that the data represents some aspect of reality, the strategy for seeking out implicit or hard-to-perceive relationships should make sense within the specific context of exploration.

The data exploration environment can be categorized as either "supervised" or "unsupervised". In a supervised learning environment, a human analyst supplies the exploration tool with metadata, or information that describes the various data attributes that are found in the data records. This gives the exploration tool knowledge of the environment the data is supposed to describe. With this data, an exploration tool can determine the appropriate set of tools that it can validly apply to the data. An unsupervised learning environment exists when no metadata is supplied. In this case the exploration tool must examine the data using the widest variety of tools, though the validity of the application of these tools to the database is up to a human analyst to determine. This mode of learning is valuable when minimal knowledge concerning the nature of the data is available.

Consider, for example, a database containing numerical data attributes. If nothing is known about this data, a numerical induction system such as BACON would apply its analysis heuristics in an attempt to determine wheher the terms are numerically related. This will very often provide an analyst with valuable information regarding the relations between terms. As stated earlier,

BACON has discovered and "rediscovered" many valuable numeric laws. However, if the data represents attributes such as a zip code, a year-of-birth, a social security number, or a yearly income, any numerical relationships discovered that relate these terms has no meaning in the real world. The purpose of data exploration systems is just the opposite: to discover trends that can be used to make generalizations or predictions about the real world. Even when the numerical data found in a database does lend itself to numerical analysis, the nature of these "measurements" must be known in order to perform valid analyses. This is discussed in more detail in Section 6.

Data collection issues are key to data exploration as well. In many instances, large volumes of historic data exist that can be readily applied to data exploration. However, many situations exist in which data is constantly being collected. Consider the amount of data that is constantly being transmitted regarding the status of the many subsystems aboard the space shuttle. If a data exploration system is to be used to detect patterns and correlations for one-time-only analysis, then this is not an important consideration. Other applications however, will require the constant digestion of data streams that describe a real-time environment. A data exploration tool should be able to analyze a fixed sample or to accept incoming data and continuously modify the detected patterns to reflect the current information (i.e., sequential analysis strategy). Without this ability, an accurate determination of the significance of the various attributes being collected cannot be made.

5. INTERPRETING the RESULTS of DATA EXPLORATION SYSTEMS

Data exploration systems can serve as a powerful instrument enabling an analyst to perceive structure in a seemingly dense forest of data. Ideally, the derived perception of the data reflects relationships and generalizations actually present and not due to chance. The data exploration system merely enhances the analyst's perception much the same way as a telescope enhances an astronomers perception. This section outlines the technical difficulties in achieving this goal. The crux of the matter is validity. Under what conditions might a data exploration system offer invalid results? The thesis is stated simply: the interpretation of results can be complex because a statistical strategy and the semantics of numerical data can strongly influence the interpretation of the results.

For the sake of illustration, consider a simple hypothetical database adapted from [Berger] containing data regarding two identical subsystems serviced by different maintenance teams (i.e., paired observations). It will be shown that the statistical strategy can be subjective and can strongly influence the discovered structure. Thus, a data analyst cannot interpret the results without knowing precisely the details of the statistical strategy used to discover the patterns. The conclusion is that the consumer of the data should control the strategy of analysis, not the data exploration tool. Assume that before a launch, each subsystem is assigned a maintenance team at random and after the flight the performance of the subsystem is evaluated. Next to the paired observation is the outcome stating which subsystem performed best (for the sake of clarity, assume ties are not allowed). Let attribute labels 1 and 2 represent the subsystems, and let their values represent which of the maintenance teams, A or B, serviced the respective subsystem. Attribute 3 represents the post-mission evaluation of which subsystem/maintenance team performed best. The database appears in Table 2. Attributes 4 and 5 are discussed shortly.

Table 2. Sample Database

1	2	3	4	5
A	B	A	42	30
B	A	B	39	41
A	B	A	39	23
B	A	A	43	33
A	B	A	40	25

A	B	A	46	40
A	B	A	44	35
B	A	B	37	39
A	B	A	54	60
B	A	A	52	54
A	B	B	42	40
A	B	A	52	54
B	A	A	50	50
A	B	A	45	38
A	B	A	47	44
B	A	A	52	54
A	B	B	38	39

The first issue regarding the interpretation of the results is very complex: what are the assumptions of the statistical strategy, and given that these assumptions hold for the data, how do these assumptions influence the interpretation? To illustrate the effect of exploration technique on interpretation, two separate statistical viewpoints are assumed and their implications examined. First, consider the view of classical statistics. Classical statistics assumes a model responsible for the data prior to examination, and that any deviation from the assumed model is caused by chance. Given the sample database, consider the following strategies:

- Strategy 1 assumes that the data is generated by a binomial distribution with no difference between the maintenance teams. The probability of the outcome of 13 favorable outcomes for Team A is 0.182 and the P-value is 0.049. One is tempted to assume that there is no difference in maintenance teams and that the pattern of 13 favorable outcomes for team A is due to chance.

- Strategy 2 assumes that the data is generated by a negative binomial using a sequential sampling plan: (i.e., data was collected until both Team A and Team B both had 4 favorable outcomes). It so happened that the last favorable outcome for Team B occured on trial 17. In this scenario the probability of the pattern of 13 favorable outcomes for Team A is 0.0085 and the P-value is 0.021. The conservative judgement is that the pattern in favor of Team A is not due to chance. The "discovery" is that Team B needs training.

What is the cause of this ambiguity? The ambiguity is caused by assuming a statistical hypothesis that in turn results in P-values for values that are unobserved yet theoretically possible.

Which of these interpretations is true and how can an analyst communicate the sampling plan and assumptions to the data exploration tool? More importantly, does the data discovery tool accommodate such metadata prior to exploration? From the above example, one can conclude that the sampling plan and assumed distribution strongly influence the interpretation of results. These assumptions are inherent in the classical statistical approach. The two distributions are sketched in Figure 3 for comparison. The leftmost distribution results when a coin is flipped 17 times and the probability of the number of heads is calculated. The rightmost distribution results when a coin is flipped until four tails appear, the last tail occurring on toss 17. The P-values are also indicated in the shaded areas; these indicate the "confidence level" of the null hypothesis.

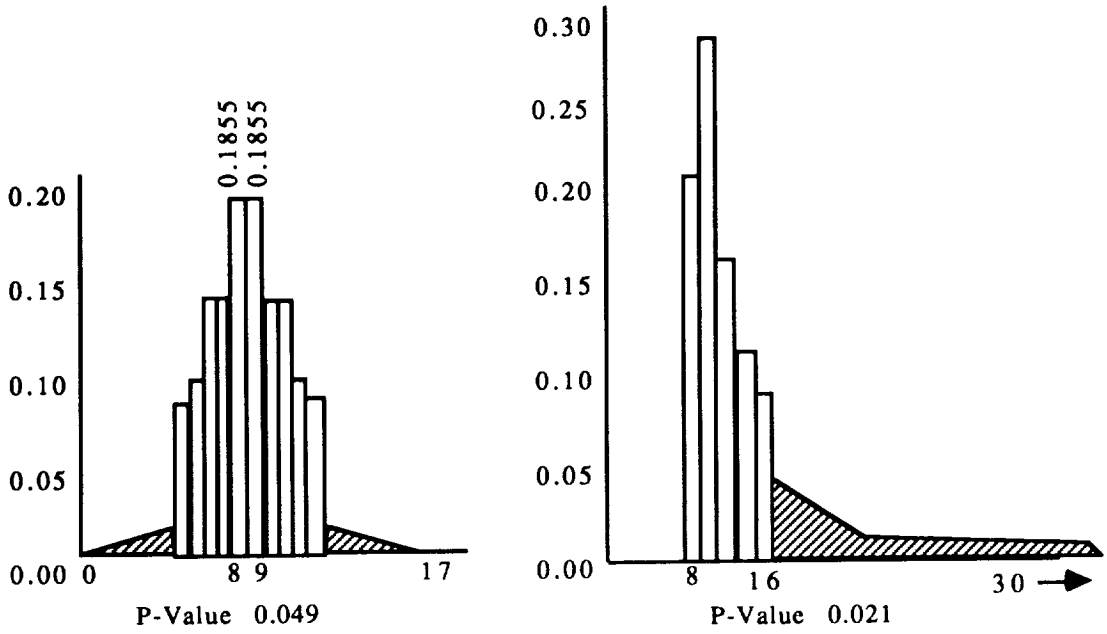


Figure 3. Two Statistical Models for Sample Database

The role of P-values is fundamental in the classical approach to data analysis, yet it is often misunderstood. P-values consist of both observed data and hypothesized (unobserved) data; these represent the probability that the null hypothesis is falsely rejected *assuming that the null hypothesis is true.* P-values represent the probability of obtaining data that casts at least as much doubt on the hypothesis as on the observed data itself. However, the P-values do not indicate the truth of the null hypothesis in the face of the data. P-values essentially give indirect evidence against the null hypothesis.

An alternative way of exploring the data is a Bayesian approach. On the surface, the Bayesian approach seems more appropriate to data exploration. First, the Bayesian approach allows the formal integration of prior knowledge into the hypothesis generation process. Next, the Bayesian approach permits sequential sampling and the accumulation of evidence. The ability to accumulate evidence and make decisions when the evidence becomes strong enough is appropriate to many experimental engineering analysis problems. The classical approach requires sample sizes to be specified in advance and the sampling conditions to remain uniform for the duration of the sampling. Then the data is analyzed. However, this type of sampling and decision regimen may not be suitable for the analysis of real-world, mission-critical operations. More appropriate for hypothesis generation is the integration of the best engineering judgement into the analysis of the arriving data and making the best decisions possible given the data at hand. Ideally, the set of hypotheses can be scored and the most probable hypothesis selected based on direct evidence plus prior knowledge. Classical data exploration techniques are of limited value in this situation. Bayesian techniques on the other hand, directly estimate the truth of a hypothesis given the data. In short, the Bayesian approach addresses itself directly to the issue of how degrees of belief are to be altered by the *observed* data. Contrast this approach with the concept of P-values, which account for *unobserved* data. Bayesian techniques are unaffected by unobserved data and thus permit a sequential sampling plan; in addition Bayesian techniques are not invalidated by nonuniform sampling conditions or affected by experimental design. Data exploration in mission-critical contexts can benefit from a Bayesian approach, yet few data exploration systems fully accommodate this approach.

6. HOW the SEMANTICS of NUMBERS AFFECTS INDUCTION and GENERALIZATION

As discussed earlier, a key issue in data exploration is the amount of domain specific knowledge necessary to apply mathematical pattern detectors. This again touches on the nature of supervised learning. A tentative answer is that data exploration does require at least a "modest" amount of semantic metadata. To see this, consider the two integers 20 and 45. If their mean value is to be used for statistical purposes, one might use the arithmetic mean to obtain an average of 32.5. However, if these integers represent a "rate" such as kilometers per hour, then the mean value should actually be 27.77 (assuming an equal weighting). When determining the mean of rates involving unit ratios (miles/hour, ohms/meter, etc.), the *harmonic* mean must be used to determine the true mean. For example, if a vehicle goes one kilometer at 20 km/hr and the next kilometer at 45 km/hr, the average velocity is 27.77 km/hr, not the 32.5 k/hr that the *arithmetic* mean would indicate. Now, if this mean is to be used to calculate other values, such as times of arrival given certain time intervals, this difference in velocity will quickly cascade through further calculations, most likely unnoticed. Though simple, this example demonstrates that all numeric data cannot be handled uniformly. A data exploration system must have some metadata describing what the numbers represent.

One important shortcoming of most data exploration systems is that numerical values *are* treated uniformly. Numbers represent some aspect of reality and the results of numerical operations are assumed to make true statements about this reality. However, our research indicates that, in general, numbers cannot be treated uniformly and mathematical operations cannot be applied without empirical justification. Specifically, if a user cannot communicate the semantics of numerical values to the data exploration system, the corresponding results may be suspect. "Measurement theory" offers important design guidelines detailing the qualities of measurements that can be used to avoid the above problems.

Consider attributes 4 and 5 in the sample database in Table 1. If numerical values are treated simply as "numbers", then the pattern "when attribute 3 = 'A', then the value of attribute 4 = 2/5(attribute 4) + 30" might be detected. Is this valid? At first glance it seems that, indeed, this generalization is true. Yet further examination demonstrates that the validity of this generalization depends on the semantics of the numbers themselves. First, assume that the numbers represent the payroll numbers of the two supervisors of Team A and Team B. The generalization in this case is probably senseless. However, if the two attributes represent outside temperature and the temperature of a subsystem, the generalization is valid only within the scale on which the temperatures were measured. In other words, the generalization is invalid if, in the future, temperature is measured in Fahrenheit instead of Centigrade or vice versa. The co-ordinate system itself induced the generality. If we change the co-ordinate system, the generalization disappears. However, if the attributes represent the weight of two related subsystems, then the generalization is valid regardless of the co-ordinate system in which the original measurements were made. The reasons for these assertions are grounded in measurement theory.

Briefly, measurement theory is a branch of mathematics that formalizes the practice of associating numbers with objects and empirical phenomena and the interpretation of those numerical values. Numbers can take four different meanings, and these meanings constrain the types of operations that result in valid application. First, a number can be "nominal". This means the number represents a qualitative symbol such as a name. In the example above, employee number is nominal. Next, a number can be "ordinal". This means the number represents a location in a ranking but not magnitude (e.g., the object ranked fourth does not necessarily have twice the ranked property as the item placed second even though (2)*2 = 4). Third, a number can be a "ratio" measurement. This means that the number represents a measurement with an arbitrary scale and origin. In the example above, temperature is such a measurement. Note that if temperature X is twice temperature Y in Fahrenheit, this is not necessarily true if the numbers are converted to

Centigrade. Finally, a number can represent an "interval" measurement. This means that the property measured has a "natural" absolute origin. In the example above, weight is such a measurement because at zero G, the object measured has no weight regardless of the specific measuring scale. A table of allowable transformations is given below.

Table 3. Valid Operations Based on Measurement Type

Type of Measurement	Admissible Transformation	Example
nominal(symbolic)	any 1-1 transform	numbers used as labels
ordinal	x >= y iff f(x) >= f(y)	preference rankings
interval	f(x) = ax + b	temperature in C or F
ratio	f(x) = ax, a > 0	weight

Table 3 indicates that the generalization "X is cY where c is constant" is an invalid operation on interval measurements such as temperature because generalizations induced in one temperature scale do not necessarily hold in another scale. The admissible transformations for a measurement type are the "litmus test" for the validity of an inductive generalization. See the Appendix for proof of this assertion.

The interested reader is referred to [Roberts] for details, implications, and formal proofs about the validity of numerical inferences. The importance of measurement theory to data exploration cannot be overstated. This theory shows that numerical values cannot be treated uniformly, because the results can be invalid. On the positive side, measurement theory also indicates that data exploration systems need not possess a great deal of domain specific knowledge. All that needs to be insured is that the appropriate transformation is applied to a numerical value. If the dictums of measurement theory are obeyed, then the opportunities for invalid generalization are reduced.

7. INTEGRATION of SYMBOLIC and NUMERIC DATA

A related issue is the integration of symbolic and numeric data. Most systems use one type of data exclusively and fail to exploit the information carried by both types. Yet, many databases contain both types. A single coherent computational framework is needed for using both qualitative and quantitative data in searching for potentially meaningful patterns. One framework for integrating both symbolic and numeric information is to cluster the numeric information, assign symbolic cluster names, and use the cluster names in an algorithm such as ID3, which discovers low entropy attributes with respect to a given taxonomy. This technique was tried on our prototype system using a simple Euclidean distance measure and the *maximin-distance* clustering algorithm. The key issue in this approach is to make an informed guess about the numeric data and select a suitable distance measure. Once again, a modest amount of domain-specific knowledge must be applied. If one knows nothing at all about the data as shown in this section, the results of any exploration approach must be suspect.

Cluster analysis organizes data by uncovering underlying structure in data either as a grouping or a hierarchy of groupings. The analyst can use the grouping as confirmatory evidence of suspected structure or as fertile ground for further experimentation to explain the discovered taxonomy [Everitt]. The AUTOCLASS program mentioned in the introduction is based on cluster analysis. An important theoretical consideration for employing cluster analysis is that it is free of the ambiguities induced by P-values and other statistical assumptions previously discussed. Thus, cluster analysis can serve as both a remedy for the problems of classical or Bayesian statistical methods or as an additional validation technique to supplement these methods. The database of Table 2, when clustered, appears in Figure 4:

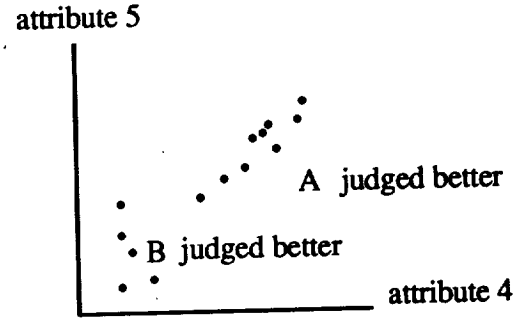


Figure 4. Clustering

One hypothesis from the clusters might be that Team A actively trains, while Team B is shamed into training also. When Team A does not train, Team B relaxes and then wins on talent alone.

There are several issues involved with cluster analysis. First, how are clusters defined? Exactly what shapes and distances define a cluster are domain dependent parameters and are often unknown before analysis. Next, research suggests that clustering without some domain knowledge is still a weak technique. The reason is that clusters are defined by distance measures that themselves have meaning within a domain. The Euclidean distance is just one example of a potential distance measure useful in clustering. Finally, which attributes should serve as the clustering attributes? These all depend on the goals of the analysis. Consider the different ways of clustering a deck of cards: one may form clusters based on numeric value, suit, color, etc. The data exploration system should provide a mechanism for defining distance measures and allowing the user to search for structure based on different grouping criteria.

Our experiments use clustering and distance measures not only as stand alone techniques (as in the example) but in data reduction as well. Clusters of numeric data are tagged with names, and the names serve as an additional attribute that represents and classifies a group of the numeric values. Then a symbolic induction technique such as ID3 finds those factors correlated with the cluster. A similar technique is used for curve fitting. When a set of numeric data is fitted with a curve, the coefficients of the curve are compared with the curves fitted to related data. Using the Chi-Square Test, a distance measure is defined, and a determination regarding the two sets of numeric data is made. If the data are close, they are grouped within the same cluster. The same maximin algorithm can be applied with a different distance measure.

8. CONCLUSIONS AND RECOMMENDATIONS

Our research suggests several conclusions. First, data exploration is based on the detection of regularities in data. Therefore, the data exploration tool should provide a variety of detection techniques for the types of regularity likely to occur in the data. Without this, the system is blind to potentially important and characteristic patterns. Next, symbolic data patterns can be detected using a variety of statistical techniques. Various entropy measures have been shown to be useful in this endeavor. Decision trees constructed from the output of algorithms such as ID3 based on entropy measures offer a visual representation for the structure of the data. Third, data exploration on numeric values is very complex due to the semantics of numbers. Care must be taken to avoid transforming the data in meaningless ways and deriving invalid patterns on the data. Finally, more research is needed into integrating symbolic and numeric data into a coherent framework for data exploration.

Several recommendations are made for future directions for research. Some enhanced data exploration techniques have already been implemented and are undergoing experimentation.

Experiments are underway to integrate symbolic and various types of numeric data into an overall methodology for data exploration. Promising techniques include the incremental integration of domain-specific knowledge into cluster analysis and curve-fit analysis.

The role of conjecture in the discovery process is well-recognized [Polya]. As stated previously, data exploration tools can support the formulation of a conjecture by examining the data and elucidating the "structure" of the data. Structure in the sense used here means "regularity" or generalization exemplified by the data. Regularities are described and hence detected mathematically. The user must verify that the assumptions underlying the application of the mathematical technique are satisfied, and if so, then the data exploration tool can perform a great deal of statistical analysis and uncover the structure of the data. Certainly, powerful tools such as multivariate statistics and information theory can provide the data exploration tool with a sturdy vehicle for exploration.

However, a paradox arises. On one hand, an analyst applies a data exploration tool *because* so little is known about the data. But, on the other hand, the very exploration techniques require a certain amount of knowledge about the data before the results can be validated. In other words, if a data exploration tool presents us with a host of conjectures, what can be said about the potential validity of the conjectures, all else being equal? Ideally, one does not need a tool to manufacture blind alleys, smokescreens, and distractions for the analyst.

What is needed is an incremental approach to integrating domain knowledge as it is acquired into the exploration process. This paper has shown that induction and generalization over a database cannot be completely free of domain knowledge. At a minimum, the data exploration system must account for the semantics of numeric data. Next, classical statistical methods must be employed with great care because both the statistical hypothesis and the method of data collection strongly affect the validity of induction and the interpretation of the resulting generalization. Finally, data exploration systems can be structured in two ways: when little is known about the domain being explored, the system should be a loosely coupled set of tools supported by metadata. The more known about the domain, the less exploration is needed, the more predictable the analysis becomes, and the more domain specific knowledge should be infused into the analysis process. The tool supporting the analysis should be flexible enough to accommodate a variety of data and analysis strategies.

APPENDIX

Proof that the generalization of f(a) = cf(b) is invalid for interval measurements (e.g., mass, temperature on F or C, etc.) where $c > 0$.

Proof: Assume f(a) = cf(b) for some a,b,c where f(x) is a quantity assigned to x on an interval scale. The generalization is valid iff it is invariant under all admissible transformations. Since f(x) is assumed to be an interval measurement, let g(x) = kx + b where $k,b > 0$, the general admissible transformation for interval scales. If f(a) = cf(b), then (g*f)(a) = c[g*f](b) --> g(f(a)) = c[g(f(a)]. But, k f(a) + b <> c[kf(x) + b]. Hence f(a) = cf(b) is an invalid generalization in an interval scale. Note, however, that if b = 0 (i.e., the measurement type is ratio such as weight), then the generalization is valid.

REFERENCES

Berger, J.O. and Berry, D.A. (1988) Statistical Analysis and the Illusion of Objectivity. *American Scientist* , 76(2)

Denning, P.(1989) Bayesian Learning, *American Scientist* , 77(3)

Everitt, B. *Cluster Analysis* . Wiley and Sons, New York, N.Y., 1974

Hamming, R.W. *Numerical Methods for Scientists and Engineers.* Dover Publications, Inc., New York, N.Y., 1973

Hand, D.J. Emergent Themes in Statistical Expert Systems in *Knowledge, Data, and Computer-Assisted Decisions*, Springer Verlag, Berlin 1989
Hoaglin,D.C., Mosteller, F.P., and Tukey , J.H. *Exploring Data Tables, Trends, and Shapes*, Wiley and Sons, New York, N.Y., 1985
Langley,P. and Zytkow, J.M. (1989) Data-Driven Approaches to Empirical Discovery.*Artificial Intelligence* 40
Parsaye, K. ,Chignell, M, Khosafgarian,S., and Wong, H. *Intelligent Databases*, Wiley and Sons, New York, N.Y., 1989
Polya, G. Mathematical Discovery, Wiley and Sons, New York, N.Y., 1981
Roberts, F.S. *Measurement Theory* Addison-Wesley, Reading, Mass., 1979

Logic Programming and Metadata Specifications

Antonio M. Lopez, Jr., Ph.D.*
Mathematical Sciences
Loyola University Box 51
New Orleans, LA 70118
(504) 865-3340
e-mail: lopez@loynovm.bitnet

Marguerite E. Saacks, Ph.D.*
Computer Science Department
Xavier University
New Orleans, LA 70125
(504) 483-7456
e-mail: saacks@comus.cs.tulane.edu

Abstract

Artificial intelligence (AI) ideas and techniques are critical to the development of intelligent information systems that will be used to collect, manipulate, and retrieve the vast amounts of space data produced by "Missions to Planet Earth." Natural language processing, inference, and expert systems are at the core of this space application of AI. This paper presents logic programming as an AI tool that can support inference (the ability to draw conclusions from a set of complicated and interrelated facts). It reports on the use of logic programming in the study of metadata specifications for a small problem domain of airborne sensors, and the dataset characteristics and pointers that are needed for data access.

Introduction

The National Aeronautics and Space Administration (NASA) is on the verge of a tremendous data explosion. By the end of this decade, the Earth Observing System (EOS), just one of NASA's projects, is expected to produce several terabytes of archival data each week. These data will be in a variety of formats and will "belong to" a variety of Earth and Science disciplines.

Although mass storage device technology, which makes megabyte data files practical and affordable, is keeping pace with current industrial and business demands, new innovative software systems will be required to organize, link, maintain, and properly archive the EOS data that is to be collected for the EOS Data and Information System (EOSDIS) (Dozier, 1990). Software problems associated with organizing, structuring, and managing these very large multi-format data files for efficient and timely access and update are being addressed. Artificial intelligence tools, techniques, and concepts offer great potential in solving many of the software problems that have already surfaced.

The Intelligent Data Management (IDM) project team at NASA's Goddard Space Flight Center (GSFC) is developing a prototype system for managing the terabytes of satellite imagery data that EOS is expected to produce. The research and development incorporates a number of

---

* This work was begun when the authors were 1991 NASA/ASEE Summer Faculty Researchers in the Information Systems Division of the Science and Technology Laboratory at the John C. Stennis Space Center in Mississippi.

state-of-the-art AI software methodologies in an effort to provide new insights and tools for building future intelligent information systems (Campbell and Cromp, 1990). Published works by members of the IDM project team discuss a high-level expert system for declarative and procedural knowledge acquisition (Cromp, 1988), an intelligent user interface for browsing satellite data catalogs (Cromp and Crook, 1989), the application of connectionism to query planning and scheduling (Short and Shastri, 1990), and an architecture for a large object-oriented database (Dorfman, 1991). At the heart of the work that is being done at GSFC is the Intelligent Information Fusion (IIF) concept, a structured approach to implementing the management and access to data, metadata (useful information about the data), and supporting information and knowledge (Roelofs and Campbell, 1990). An essential element of IIF is the semantic and knowledge-based representation that captures the essence of the data domain at all levels of knowledge representation, from the highest class structure, to the intermediate metadata, to the lowest level of data granule. The overall concept of implementing an Intelligent Information Fusion System (IIFS) for spatial data management has been described by Campbell et al. (1990).

An AI Tool

Logic programming is an outgrowth of the research that was done in the mid-1960's on automated inferencing and theorem proving. A logic program is constructed by describing what is true in a particular problem domain. It is equivalent to a set of logical axioms. These axioms are facts and rules that describe objects and the logical relationships between them. The execution of a logic program is equivalent to a constructive proof of a goal statement from the axioms, and it is carried out by an application-independent inference procedure (Genesereth and Ginsberg, 1985) embedded within the particular programming language implementation.

Logic programming provides an efficient mechanism to integrate data, metadata, and control into a domain-specific knowledge environment (Kerschberg, 1990). Recently, logic programming languages such as LDL (Naqvi and Tsur, 1989) and LOGIN (Ait-Kaci et al., 1990) have been designed for efficient access to very large collections of data and for using concepts such as inheritance. Both of these languages are extensions of PROLOG (PROgramming in LOGic), the flagship of logic programming languages.

PROLOG has been shown to be a useful AI tool in a wide range of applications such as expert systems (Moller-Jensen, 1990), relational databases (Lucas, 1988), knowledge representations (Goyal, 1989), and natural language processing (Tanaka, 1988). More recently, attention has been drawn to PROLOG as a specification language (Denney, 1991), and for use in declarative testing and debugging (Yan, 1991). It is these two aspects of this logic programming language that we intend to exploit in metadata specification.

The Theory

Metadata provides systems such as EOSDIS and IIFS with a knowledge model that captures the data semantics (i.e., objects, properties, etc.) and the knowledge semantics (i.e., heuristics, scripts, etc.) of a particular domain. The truly creative and most difficult step in the development of metadata is the construction of an acceptable formalism from an intuitive understanding of the data domain, using design tools such as semantic networks, frame-based representations, and object-orientations (Cercone and McCalla, 1987). In specifying the metadata, the intelligent information system developers claim to know: (1) what knowledge to represent in an application, and (2) how to reason with that knowledge. Regardless of the tool used to specify the metadata, the question is, "Does the metadata provide an accurate knowledge model?"

In education theory, the deductive model of inquiry treats theories as: (1) a set of basic facts

and principles, and (2) a deductive logic that allows explanations and predictions to be derived. McEneaney (1990) has shown that there is a clear connection between theories in the deductive model of inquiry and logic programming. Logic programs can be used not only to test the validity of theoretical arguments, but also to make substantive contributions to theory development and revision. Logic programs can also be used to develop a metadata specification, because metadata is a theory about the relationships that exist among the data.

PROLOG can be used as an AI tool to construct and test metadata specifications given in terms of a semantic network, a frame-based representation, or an object-orientation. Specifications written in PROLOG are executable. Because PROLOG makes no distinction between data and program, it is a powerful tool for simulating the learning needed in intelligent information systems. In addition, this approach allows the developer to "ask questions" of the metadata, derive answers, and change the metadata if the answers are unacceptable. Through an iterative process of generate and test, the metadata specifications and the PROLOG program must eventually produce an accurate knowledge model.

The Application

NASA's John C. Stennis Space Center (SSC) has over the years collected data obtained by using the Thermal Infrared Multispectral Scanner (TIMS), and the Calibrated Airborne Multispectral Scanner (CAMS). The analog tapes produced by these sensors on different missions are stored in SSC's data holdings and digitized for Earth Scientists. In anticipation of EOSDIS and IIFS, the Information Systems Division at SSC was interested in developing metadata specifications for their TIMS and CAMS data sets.

An initial investigation revealed two important points. First, the information that was stored on the tape headers was not enough to support the types of queries that scientists in the Earth Science Division would want to make. For example, scientists suggested queries that needed information found on the Mission Flight Request Form, a five page document with possible attachments. The Mission Flight Request Form is not stored electronically with the data that was obtained by the mission. Second, people's understanding of metadata varied. Some proposed tables that could be implemented using a relational database management system; others produced the *NSSDC Directory Interchange Format Manual (Version 3.0, December 1990)* and indicated that a directory entry consists of collections of "metadata" fields; and yet others knew the purpose of metadata, but found it difficult to specify.

We decided to view metadata as a theory about the underlying data sets. Given facts and principles, the metadata would be used to "predict" the need for a particular data set. Viewed in this way, metadata would be analogous to a theory in the deductive model of inquiry, and it would be reasonable to build the theory as a logic program that would be analyzed, tested, and revised.

Two of the most successful approaches to building knowledge representation systems have been semantic networks and frames. One advantage of semantic networks is the simplicity with which logic can be used to answer questions. Frames have proven invaluable in organizing large numbers of facts. Both of these knowledge representation approaches were used to specify the requirements for the TIMS and CAMS metadata (Saacks and Lopez, 1992). Metadata was organized as a semantic network using explicit relationships between objects. Complex objects were represented as a single frame instead of a larger network.

An abbreviated portion of the semantic network developed to specify the metadata for the TIMS and CAMS data holdings at SSC is given in Figure 1. The data set pointer objects (**ssc100, ssc110, ..., ssc180**) are stacked and associated with the TIMS or CAMS sensor

that created it. This is done only for the convenience of presentation. Figure 1 shows that the **sensor** object inherits from both the **tool** object and the **platform** object. This is an instance of multiple inheritance. Similarly, the data set pointer objects inherit from the **flight** object, and from either the **tims** or **cams** object. What Figure 1 does not show is the complexity of each object.

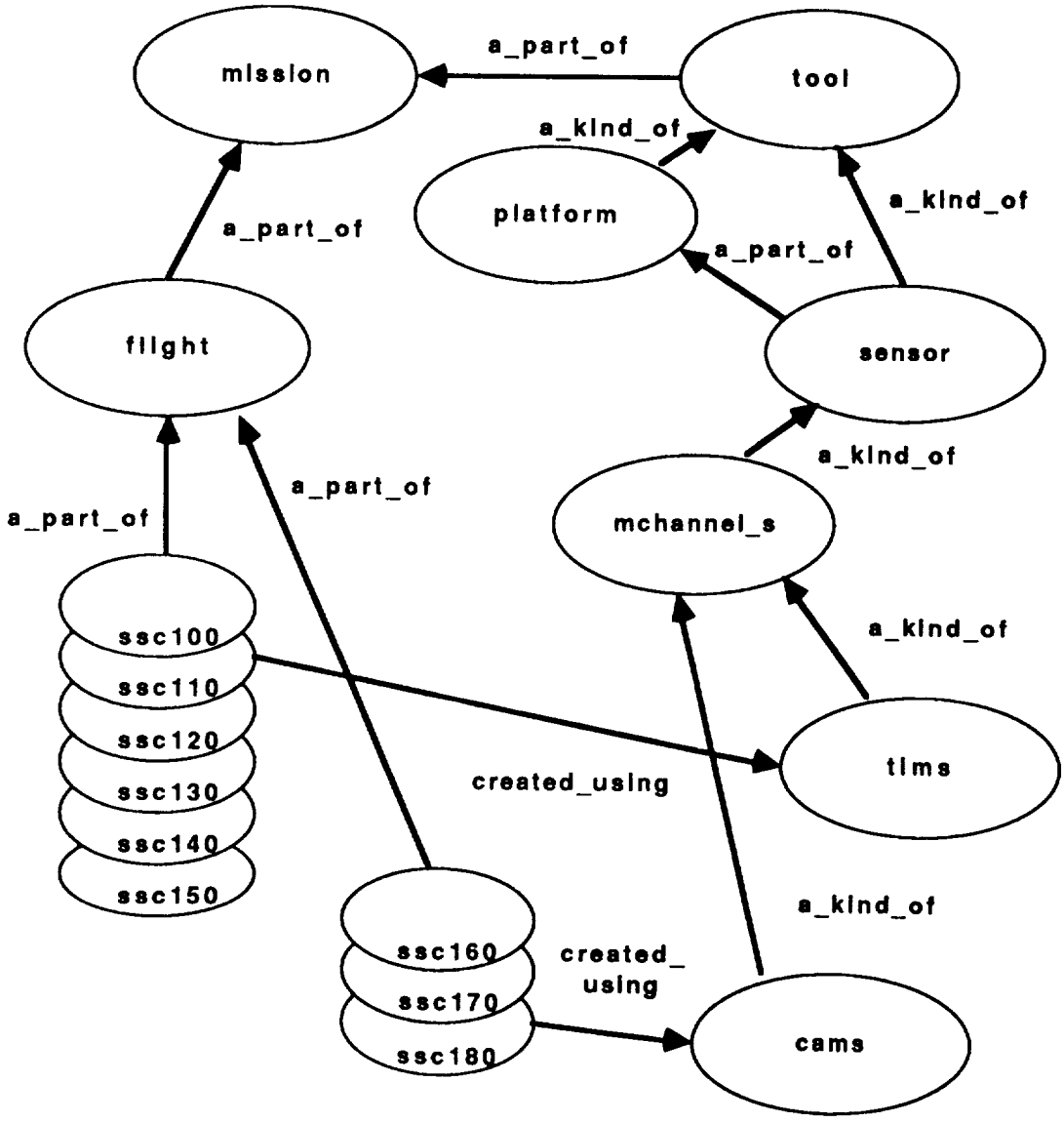


Figure 1. Metadata as a semantic network.

Figure 2 takes some of the objects in Figure 1 and develops them as frames with both unfilled and filled slots. This shows the complexity of the objects as well as the concept of slot inheritance. For example, all multichannel sensors (**mchannel_s**) have a **resolution** slot but it is not filled unless there is a specific data set pointer. Since the **mchannel_s** frame has a **resolution** slot, the **tims** frame, which is **a_kind_of mchannel_s**, has it, too.

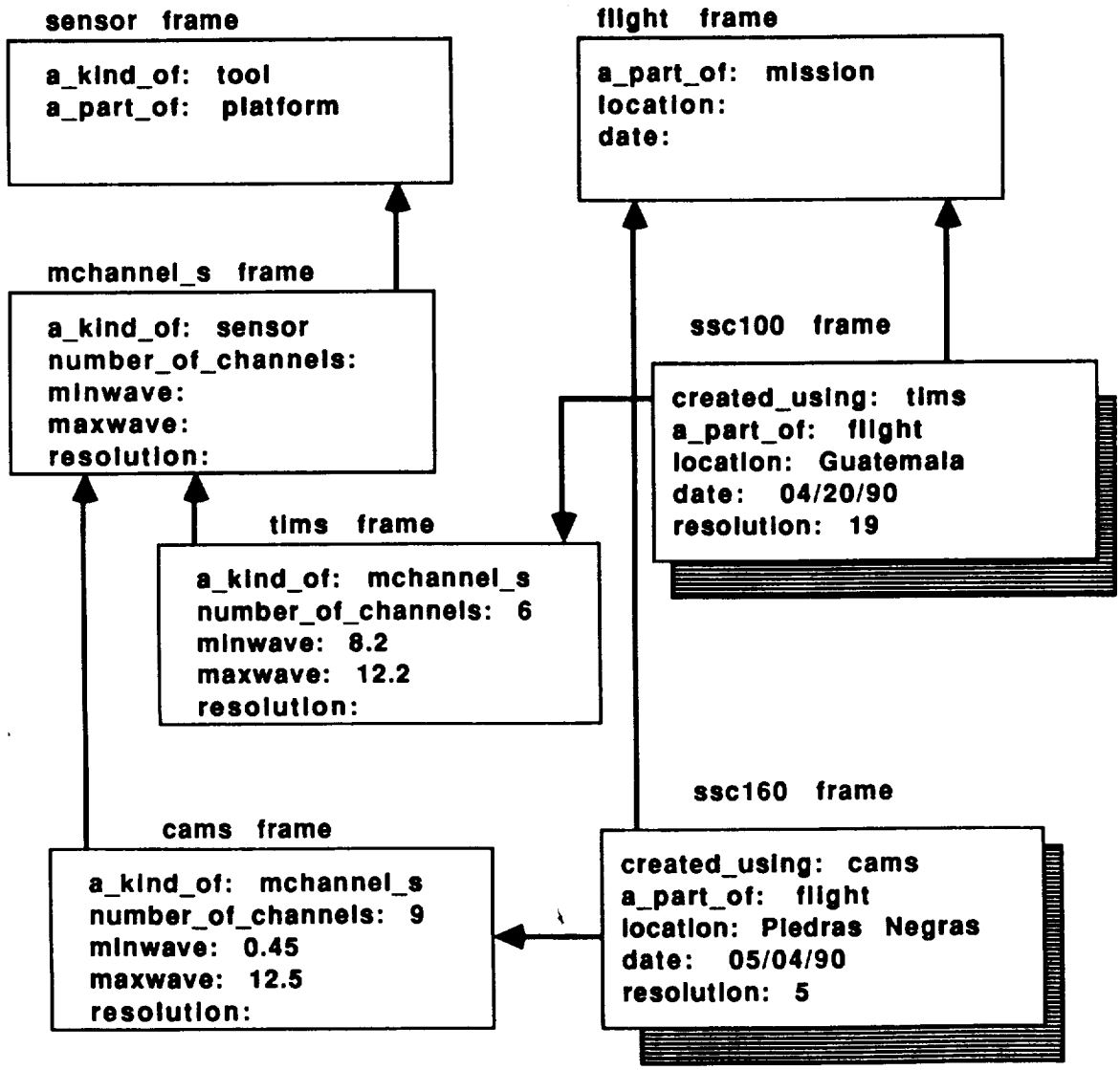


Figure 2. Metadata as frames.

The semantic network and frames indicated facts and principles that had to be represented in the metadata. However, we still needed to know if we could reason with this knowledge. Could the metadata provide a deductive logic that would allow predictions to be derived about what data to retrieve? The work of McEneaney (1990) suggested the use of logic programming to address this question.

Taking the frames, we coded the metadata into PROLOG. For example, the **flight**, **mchannel_s**, and **tims** frames became the following:

```
% Flight Frame ----------------------------------------

value(flight,a_part_of,mission).

slot(flight,location).
slot(flight,date).


% Multichannel Sensor Frame ---------------------------

value(mchannel_s,a_kind_of,sensor).

slot(mchannel_s,number_of_channels).
slot(mchannel_s,minwave).
slot(mchannel_s,maxwave).
slot(mchannel_s,resolution).

units(mchannel_s,minwave,micron).
units(mchannel_s,maxwave,micron).
units(mchannel_s,resolution,meter).


% TIMS Frame ------------------------------------------

value(tims,a_kind_of,mchannel_s).
value(tims,number_of_channels,6).
value(tims,minwave,8.2).
value(tims,maxwave,12.2).
```

Note that the **slot** predicate is used for those slots that are unfilled, while the **value** predicate is used for those slots that are filled. This approach makes writing inference rules simpler. Also, since slots having numeric values can have associated information, say about the units of measurement, we have included a **units** predicate.

The data set pointer frames here require the use of the **value** predicate only. However, in a completely developed metadata system, the frame would contain the rules by which the underlying data could be retrieved (i.e., E-mail addresses, login accounts, telenet machine numbers, etc.). The frame name would be the "entry_id" as defined in the *NSSDC Directory Interchange Format Manual*. For our prototype, the data set frame names are keys that we want our metadata to predict. Some examples of data set pointer frames written in PROLOG are:

```
% Data Set Pointer Frames ---------------------------------

value(ssc110,created_using,tims).
value(ssc110,a_part_of,flight).
value(ssc110,location,'Site 1/Peten').
value(ssc110,date,'04/21/90').
value(ssc110,resolution,5).

value(ssc120,created_using,tims).
value(ssc120,a_part_of,flight).
value(ssc120,location,'Site 1/Peten').
value(ssc120,date,'04/22/90').
value(ssc120,resolution,5).

value(ssc130,created_using,tims).
value(ssc130,a_part_of,flight).
value(ssc130,location,'Piedras Negras').
value(ssc130,date,'04/23/90').
value(ssc130,resolution,5).
```

The inference rules that can be applied to these frames can be written independently of the particular application. To be able to test and debug the metadata specification, we need value inheritance rules, slot inheritance rules, rules enabling qualifying slots to be inherited, and a good deal more. An example of the slot inheritance rule is:

```
has_slot(Object,Slot) :- slot(Object,Slot).
has_slot(Object,Slot) :- value(Object,a_kind_of,Superclass), has_slot(Superclass,Slot).
```

It should be mentioned at this point that the principle control mechanism of the "standard" PROLOG interpreter is depth first searching. Since PROLOG is an extensible language, other control strategies may be substituted.

The following are some examples of queries to and responses from the PROLOG code:

```
?- has_value(tims,Metadata_slot,Slot_value).
Metadata_slot = a_kind_of                Slot_value = mchannel_s
Metadata_slot = number_of_channels       Slot_value = 6
Metadata_slot = minwave                  Slot_value = 8.20
Metadata_slot = maxwave                  Slot_value = 12.20
Metadata_slot = a_kind_of                Slot_value = sensor
Metadata_slot = a_kind_of                Slot_value = tool
Metadata_slot = a_part_of                Slot_value = platform

?- has_slot(What,minwave).
What = mchannel_s   What = tims   What = cams

?- has_value(Entry_id,created_using,tims), has_value(Entry_id,location,'Site 1/Peten').
Entry_id = ssc110   Entry_id = ssc120
```

?- has_slot(Frame,Something), has_value(Entry_id,Something,'Piedras Negras').
Frame = flight Something = location Entry_id = ssc130
Frame = flight Something = location Entry_id = ssc160

?- has_value(ssc110,created_using,Sensor),has_value(Sensor,minwave,Minwave),
has_units(Sensor,minwave,Units).
Sensor = tims Minwave = 8.2 Units = microns

The first query demonstrates a browse of the TIMS frame. The responses reveal the filled slots in the TIMS frame, as well as the filled slots in the Multichannel Sensor and Sensor frames, since the TIMS frame inherits those values. The second query looks for those frames that have a particular slot, filled or unfilled. The third query is the command, "Give the key for any TIMS data set on Site 1/Peten." This is a more complex query than the previous ones in that it involves two constraints on the data set pointer frames. The fourth query is an example of partial information obtaining a result. This query seeks to find ancillary information about Piedras Negras data sets as well as keys. Finally, the fifth query represents the question, "What is the minwave and its units for the sensor used in creating the data set with key ssc110?"

In developing the PROLOG model of the metadata, our goal was to show what would be obtained by browsing, and to verify that certain keys would be obtained when particular facts were given in a query. Hence in a very empirical manner addressing the question, "Does the metadata provide an accurate knowledge model?" Earth scientists could propose questions and we could query the PROLOG model to determine if the metadata produced usable results.

Conclusion

PROLOG is an AI tool that can be used to write and test metadata specifications. A PROLOG model of the metadata can be used to gain insights into the relationships that the metadata attempts to capture. By querying the PROLOG model built for the TIMS and CAMS data sets at SSC, we were able to confirm relationships and access paths to data set pointers. Furthermore, we gained new insights into relationships, and realized the existence of relationships that had gone unnoticed, such as the need for a **created_using** relationship. PROLOG can be used to quickly prototype metadata before it is embedded in an intelligent information system, thus saving time and money by insuring that needs are met. Furthermore, since PROLOG is at the heart of logic programming languages such as LDL and LOGIN, it is conceivable that the work done with metadata specification can flow directly into an intelligent information system designed for accessing very large collections of data.

References

Ait-Kaci, H., Nasr, R., and Seo J. (1990). Implementing a Knowledge-based Library Information System with Typed Horn Logic. *Information Processing and Management,* 26(2), 249-268.

Campbell, W. and Cromp, R. (1990). Evolution of an Intelligent Information Fusion System. *Photogrammetric Engineering and Remote Sensing,* 56(6), 867-870.

Campbell, W., Cromp, R., Hill, S., Goettsche, C. and Dorfman, E. (1990). Intelligent Information Fusion for Spatial Data Management. *Proceedings of the 4th International Symposium on Spatial Data Handling,* 2, 567-578.

Cercone, N. and McCalla, G. (1987). *The Knowledge Frontier: Essays in the Representation of Knowledge,* New York: Springer Verlag.

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

Cromp, R. and Crook, S. (1989). An Intelligent User Interface for Browsing Satellite Data Catalogs. *Telematics and Informatics,* 6(3/4), 299-312.

Denney, R. (1991). Test-case Generation from PROLOG-based Specification. *IEEE Software,* 8(2), 49-57.

Dozier, J. (1990). Looking Ahead to EOS: The Earth Observing System. *Computers in Physics,* 4(3), 248-259.

Dorfman, E. (1991). Architecture of a Large Object-oriented Database for Remotely Sensed Data. *Proceedings of ACSM/SPRS/Auto Carto 10.*

Genesereth, M. and Ginsberg, L. (1985). Logic Programming. *Communications of the ACM,* 28(9), 933-941.

Goyal, P. (1989). Intelligent Information Systems: The Concept of an Intelligent Document. *Information Systems,* 14(4), 351-358.

Kerschberg, L. (1990). Expert Database Systems: Knowledge/Data Management Environments for Intelligent Information Systems. *Information Systems,* 15(1), 151-160.

Lucas, R. (1988). *Database Applications Using PROLOG.* New York: John Wiley and Sons.

McEneaney, J. (1990). Logic Programming as a Theoretical Tool in Educational Research. *Journal of Artificial Intelligence in Education,* 2(1), 63-78.

Moller-Jensen, L. (1990). Knowledge-based Classification of Urban Area Using Texture and Context Information in Landsat-TM Imagery. *Photogrammetric Engineering and Remote Sensing,* 56(6), 899-904.

Naqvi, S. and Tsur, T. (1989). *A Logical Language for Data and Knowledge Bases.* Rockville: Computer Science Press.

Roelofs, L. and Campbell, W. (1990). Using Expert Systems to Implement a Semantic Data Model of a Large Mass Store System. *Telematics and Informatics,* 7(3/4), 361-377.

Saacks, M. and Lopez, A. (1992). A Frame-Based Design for the TIMS and CAMS Metadata for a Stennis Information Management System. Under review.

Short, N. and Shastri, L. (1990). The Application of Connectionism to Query Planning/Scheduling in Intelligent User Interfaces. *Telematics and Informatics,* 7(3/4), 209-220.

Tanaka, H. (1988). DCKR-Knowledge Representation in PROLOG and its Application to Natural Language Processing. In *Proceedings of the First Franco-Japanese Symposium in Programming of Future Computers,* Fuchi, K. and Nivat, M. (Eds.), 427-439. Amsterdam: Elsevier Science Publishing.

Yan, S. (1991). *Foundations of Declarative Testing and Debugging in Logic Programming.* New Jersey: Ablex Publishing.

Acknowledgements - The authors would like to thank Kirk Sharp, Gay Irby, Bobby Junkin, and Terry Jackson of the Information Systems Division (NASA/SSC), Tom Sever and Doug Rickman of the Earth Science Division (NASA/SSC), and Bill Campbell and Bob Cromp of the NSSDC IDM project team (NASA/GSFC) for their comments and support.

Call for Papers

NASA 1993

Goddard Conference on Space Applications of Artificial Intelligence

May 1993
NASA Goddard Space Flight Center
Greenbelt, Maryland

The Eighth Annual Goddard Conference on Space Applications of Artificial Intelligence will focus on AI research and applications relevant to space systems, space operations, and space science. Topics will include, but are not limited to:

- Knowledge-based spacecraft command & control
- Expert system management & methodologies
- Distributed knowledge-based systems
- Intelligent database management
- Fault-tolerant rule-based systems
- Simulation-based reasoning
- Fault isolation & diagnosis
- Planning & scheduling
- Knowledge acquisition
- Robotics & telerobotics
- Neural networks
- Image analysis

Original, unpublished papers are now being solicited for the conference. Abstracts should be 300–500 words in length, and must describe work with clear AI content and applicability to space-related problems. Two copies of the abstract should be submitted by September 1, 1992 along with the author's name, affiliation, address and telephone number. Notification of tentative acceptance will be given by September 16, 1992. Papers should be no longer than 15 pages and must be submitted in camera-ready form for final acceptance by November 16, 1992.

Accepted papers will be presented formally or as poster presentations, which may include demonstrations. All accepted papers will be published in the Conference Proceedings as an official NASA document, and select papers will appear in a special issue of the international journal *Telematics and Informatics*. There will be a Conference award for Best Paper.

No commercial presentations will be accepted

Sponsored by NASA/GSFC

Mission Operations and Data Systems Directorate

1993 Goddard Conference on Space Applications of Artificial Intelligence
May 1993 — NASA/GSFC, Greenbelt, MD

❑ ***Abstracts due:*** Sept. 1, 1992
❑ ***Papers due:*** Nov. 16, 1992
❑ ***For further info call:*** (301) 286-3150

❑ ***Send abstracts to:***
Carl F. Hostetter
NASA/GSFC
Code 531.1
Greenbelt, MD 20771

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 1992	Conference Publication

4. TITLE AND SUBTITLE
1992 Goddard Conference on Space Applications of Artificial Intelligence

5. FUNDING NUMBERS
JON-530-030-09-01-25

6. AUTHOR(S)
James L. Rash, Editor

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA-Goddard Space Flight Center
Greenbelt, Maryland 20771

8. PERFORMING ORGANIZATION REPORT NUMBER
92B00045
Code 530

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, D.C. 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA CP-3141

11. SUPPLEMENTARY NOTES
Rash: NASA-Goddard Space Flight Center, Greenbelt, Maryland, 20771

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Category 63

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This publication comprises the papers presented at the 1992 Goddard Conference on Space Applications of Artificial Intelligence held at the NASA/Goddard Space Flight Center, Greenbelt, Maryland, on May 5-6, 1992. 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 and Scheduling, Control, Fault Monitoring/Diagnosis/Recovery, Information Management, Tools, Neural Networks, and Miscellaneous Applications.

14. SUBJECT TERMS
Artificial Intelligence, expert systems, planning, scheduling, fault isolation, fault diagnosis, control, neural networks.

15. NUMBER OF PAGES
264

16. PRICE CODE
A12

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

National Aeronautics and
Space Administration
Code JTT
Washington, D.C.
20546-0001

Official Business

Penalty for Private Use, $300

NASA

NASA

National Aeronautics and
Space Administration

Washington, D.C.
20546

Postage and Fees Paid
National Aeronautics and
Space Administration
NASA-451

Official Business
Penalty for Private Use $300

U.S.MAIL

SPECIAL FOURTH CLASS MAIL
BOOK

L3 00¹ CP-3141 920302S090569P
NASA
CENTER FOR AEROSPACE INFORMATION
ACCESSIONING DEPT
P O BOX 8757 BWI ARPRT
BALTIMORE MD 21240