Expert Systems Built By The "Expert": An Evaluation of OPS5

May 13, 1987

Robert Jackson [1]
Astronomy Programs, Computer Sciences Corporation
Space Telescope Science Institute [2]
3700 San Martin Drive
Baltimore, MD 21218

Abstract

Two "expert systems" have been written in OPS5 by the "expert", a Ph.D. Astronomer with no prior experience in AI or Expert Systems, without the use of a "knowledge engineer". The first system was built from scratch and and uses 146 rules to check for duplication of scientific information within a pool of prospective observations. The second system was grafted onto another expert system and uses 149 additional rules to estimate the spacecraft and ground resources consumed by a set of prospective observations. The small vocabulary, the IF *this occurs* THEN *do that* logical structure of OPS5, and ability to follow program execution allowed the "expert" to design and implement these systems with only the data structures and rules of another OPS5 system as an example. The modularity of the rules in OPS5 allowed the second system to modify the rulebase of the system onto which it was grafted without changing the code or the operation of that system. These experiences show that "experts" are able to develop their own "expert systems" due to the ease of programming and code reusability in OPS5.

---

[1] Staff Member of the Space Telescope Science Institute

[2] Operated by the Association of Universities for Research in Astronomy for the National Aeronautics and Space Administration

1 Introduction

This paper describes the experiences of a "computer semi-illiterate domain expert" in writing two expert systems in OPS5 without the use of a Knowledge Engineer. The two systems, Duplication and Resource Usage, are described in some detail along with some of the strengths and limitations of the OPS5 language and environment.

2 Personnel Background

In order to put these "expert" systems into perspective it is useful to describe the people who created them.

The author's software experience, prior to beginning these projects, consisted of rather simple data analysis routines written in FOCAL running on a PDP 8/I and some initial exposure to the IQL database query language for a Britton-Lee IDM 500. There was no previous contact with modern programming languages, structured programming, or artificial intelligence.

The author did, however, work in the same group as a Computer Scientist who was writing a large system using OPS5 and who currently teaches OPS5 workshops. Aside from initial "hand-holding", this OPS5 expert's advice was used infrequently.

3 Scientific Duplication

The Space Telescope Science Institute is responsible for the scientific operations of the Hubble Space Telescope (HST). When launched by the Space Shuttle, the HST will allow astronomers to detect objects seven times more distant and with ten times higher resolution than from ground based telescopes. Astronomers who wish to use HST submit proposals describing the observations they wish to perform. Because of the unique capabilities of HST, much more observing time will be requested than is available. A Time Allocation Committee selects which of the proposals will be granted time on HST.

In this proposal selection process, the proposed exposures will be checked to see if any of them duplicate exposures already in the HST archives, exposures proposed by other scientists, or exposures which are "reserved" by the Guaranteed Time Observers (GTO's), the scientists who developed the HST scientific instruments. This duplication checking is intended to identify observations which waste spacecraft time or which violate the perogatives of the GTO's.

There will be about 40,000 exposures executed in each year of HST operations. With an expected oversubscription factor of about three for the observing time requested versus the time available, the duplication checking process clearly had to be automated. The author was charged with automating that process.

3.1 Specifics of the Problem

A single observation with HST contains information with the following characteristics:

- Field of View
- Spatial Resolution
- Wavelength Range
- Wavelength Resolution
- Minimum Detectable Intensity
- Time Duration
- Time Resolution

Two observations could be considered to be duplicates to the extent which the information obtained has the same values of these characteristics. Obviously, if two exposures use identical target positions, instruments, instrument settings, and exposure times, then the same data would be obtained. The complication comes from the fact that different instrument settings and even different instruments can produce similar, though not identical, data. The Duplication checker should be able to identify cases where similar data is being obtained.

The author in the capacity of "Domain expert" analyzed the instruments and the instrument settings and created a number of criteria for what constitutes a "HIGH", "MEDIUM", and "LOW" confidence duplication. The instruments and instrument settings are sufficiently different that several different criteria can produce the same confidence duplication. For example a "LOW" confidence duplication can be produced by both:

- Identical instruments and instrument settings, but drastically different exposures times.
- Completely different instruments which have similar spectral and spatial range and resolution.

The specific exposure data available for the duplication checking are:

- Target Position - Center of Field of View - 2 Coordinates
- Target Position Uncertainty - 2 Coordinates
- Configuration - The optical path, set mechanically or electronically.
- Mode - Spatial or Time parameters of the instrument
- Spectral Elements - Wavelength Range and Resolution
- Aperture - Field of View or Spatial Resolution
- Central Wavelength - Spectral Range
- Exposure Time - Minimum Detectable Intensity

From this data, for all the proposed exposures and all the past exposures, the three types of duplicate expsoures have to be identified.

3.2 Overall Approach

The problem can be decomposed into a spatial duplication and a instrumental duplication. There is no point in checking if two targets have duplicate exposures if they are 180 degrees apart on the sky. Note that this ignores the possibility that scientifically equivalent information can be obtained from different targets which are members of the same class of objects. Identifying objects as being members of the same class would require an AI system combining data from virtually all the available astronomical catalogs and literature and would be a major project in its own right. Using such an AI system would ignore the possibility that differences between objects in the same class is the very subject of the investigation.

The separation between spatial and instrumental duplication is not as clean as it might appear. Spatial duplication can depend on the spatial range of the instrument being used. For a camera with a very wide field of view, a given target could appear in two exposures at rather different positions. Additionally where the stated uncertainty in the target position may not be a good measure of the true errors in the target position, a very flexible measure of spatial duplication is needed.

Based on these considerations, the approach finally adopted consisted of:

1. Initial Search - Find groups of targets close together on the sky.
2. Instrument Matching - Find pairs of exposures in each group with "HIGH", "MEDIUM", and "LOW" confidence instrument matches.
3. Fine Position Check - Find pairs of exposures which are either
   - At statistically indistinguishable target positions.
   - Both within the instrument's spatial range.
   - Both within a user specified distance of each other.

The "Initial Search" drastically reduces the search space of the problem. It finds groups of targets that are closer than .1 degrees of each other, a distance which is much larger than the usual "Fine Position Check" distances of about 10 seconds of arc. The larger "Initial Search" distance thus requires the "Instrument Matching" be done on exposures which may not meet the "Fine Position Check". But by performing "Fine Position Check" after the "Instrument Matching", the criteria used in "Fine Position Checking" can be selected by user consistent with the user's estimate of the reliability of the target positions and uncertainties. The flexibility of the user specification of the "Fine Position Check" was deemed to be more important than the time wasted on finding irrelevant "Instrument Matching" duplications.

The other intentional limitation built into the duplication checking algorithms is fact that time variations of the target's properties are ignored. If more than one identical observation of a target is made, all the subsequent observations are ignored. While spurious duplications of time varying targets may be produced, the number of duplications found will be greatly reduced.

3.3 The Initial OPS5 Implementation

OPS5 seemed to be the appropriate tool to implement the "Instrument Matching" part of the duplication checker. The conditions for each degree of duplication can be described in English sentences and are easily expressed into a small number of OPS5 rules. Additionally, an OPS5 program could work on all the exposures in a group of targets in "parallel" without the need for looping mechanisms, thus simplifying the coding.

The computer scientist familiar with OPS5 gave the author the VAX OPS5 User's Guide [1], VAX OPS5 Reference Manual [2], and some examples of his OPS5 code. From these documents and the code samples, the general idea of data driven program flow and the **IF** *This Data Exists* **THEN** *Do These Actions* nature of OPS5 rules became apparent. The code samples illustrated the use of comments and white space and the use of English words and phrases to name rules, data structure fields, and variables.

The non-procedural nature of OPS5 was *not* a serious impediment for this particular novice. The notion of data-driven program flow is easily comprehended and implemented. The non-procedural aspect of OPS5 is actually an advantage where a function is to operate on all the available data, e.g., no DO LOOP control is needed. The OPS5 rules simply perform the operation on all the available data.

With this initial knowledge, the author started writing rules. After writing a single file with 4300 lines of OPS5 code, the author wandered over to the Computer Scientist and asked "Am I supposed to do something with this file?" Notions such as small modules and even compilation were still new to the author.

Upon compiling the file, there were only four syntax errors. The small syntax and the uniform structure of the rules allowed a neophyte to write code largely by cutting and pasting with the editor and putting different words in the same slots.

This first version relied almost solely on the data-driven notion of how OPS5 works. Not much reliance was placed on recency (testing for the most recent data) or specificity (testing for the more descriptive rule) as a way to control program execution. Instead, most of the rules tested the value of a control element which was set when the prior function has been completed. Equivalently, there was no use of the MEA strategy (placing extra weight on the first clause in a rule) and controlling program flow by chaining from one goal to another.

The OPS5 Duplication checking program performs the following operations on the data:

- Input the data from one group of exposures.
- Rename certain spectral elements and apertures to eliminate redundant names.
- Renormalize HRS Echelle central wavelength to aid in calculating the spectral range for this spectral element.
- Eliminate exposures which repeat identical exposures.
- Find "High" Instrument matches - identical instrument parameters.
- Find "Medium" Instrument matches - small differences in instrument parameters.

- Remove HRS Echelle matches with non-overlapping spectral ranges.
- Downgrade "High" or "Medium" duplications to "Medium" or "Low" if the exposure time ratios are too large.
- Find different type of "Low" Instrument matches.
- Calculate target separation for each Instrument match.
- Perform Fine Position Check for statistically indistinguishable target positions.
- Perform Fine Position Check for both targets in entrance aperture.
- Read out duplicate exposure data.

3.4 Subsequent Development

Since these humble beginnings, the code has been changed to:

- Use different criteria to determine the various degrees of duplication. The users of this Duplication checker preferred a somewhat different set of criteria than the author originally created.
- Use the MEA strategy, which gives extra weight to the first clause of the rule, with goal chaining for program control. This control method replaced the use of control flags and allowed much easier modification of the rulebase. It was no longer necessary to check which flag had been set, where it had been set, etc., when making changes to the rules or adding new rules.
- Be contained in several small files of related rules instead of one large file. Using several small files allowed more rapid compilation of the rule changes and provided a more obvious organization of the rules for subsequent maintainers.
- Replace several similar rules with one rule reading from a data table. There were many rules where the only difference was the value of certain constants in the *IF* clauses. These rules were replaced by one rule which went to a lookup table for the values of the constants. This compression of the rulebase made the code more compact, more easily modified, and more readable.
- Perform the third, "Fine Position Check", part of the duplication checking. This feature was required by the users and was easily added to the existing OPS5 rulebase.

In the debugging of this code, the following interpreter commands were extensively used:

- **@***filename* executed a file of OPS5 commands and was very useful for inputting test data.
- **RUN** *n* executed *n* rule firings and allowed single stepping through the program execution.

- **BACK** *n* stepped back *n* rule firings and allowed reinvestigating the internal state after one or more rule firings.
- **NEXT** displayed the name of the next rule set to fire.
- **CS** displayed the names of all the rules able to fire and ID number of the data which enabled them.
- **PBREAK** *rulename* halted rule firing before *rulename* fired and allowed quickly running the program to the point before a certain rule was to fire.
- **MATCHES** *rulename* displayed the ID number of the data which satisfied each clause of *rulename* and was very useful in finding why a certain rule did not fire.
- **WATCH 3** displayed all changes to the data and the rules which were enabled as each rule fired.
- **PPWM** *data description* displayed all data which met the *data description* and allowed searching the internal data.
- **SAVESTATE** *filename* saved the internal data state and the rule firing state at a point in time to *filename* and allowed restarting the program at a certain point without having to start from scratch.
- **RESTORESTATE** *filename* restored the internal data state and rule firing state to a certain condition.

3.5 Duplication Status

The duplication checker currently uses 146 rules and it is easy to modify the conditions defining the different levels of duplication. To search 52,000 exposures requires about six hours of CPU time on a VAX 8600. Extrapolating this performance to larger sets of data is difficult. Tests on subsets of the 52,000 exposures indicate that the CPU time increases more slowly than the number of duplications found and the number of exposures checked. In the mature phase of HST operations, the number of exposure which have to be checked will be so large that the duplication rulebase will have to be tuned to reduce the execution time. For now, the duplication check is performed only once each year and the execution time is acceptable.

4 Resource Usage

The committee which advises the Director of the Space Telescope Science Institute on which proposals to select is constrained by limits on the available spacecraft resources, i.e. time, data capacity, earth shadow time, etc. They need to know how much of these limited resources each proposal's observations consumes and how much of the available resources will be consumed by all the accepted proposals.

In the scientists' proposal for HST time, they specify the target positions, exposure times, instruments, instrument settings, absolute times, relative times, etc. of the observations.

The scientists do not specify, nor can they specify, the overhead times for spacecraft slewing, data readout, and internal mechanism changes or the time spent with the target occulted by the earth. The overhead times for spacecraft slewing and earth occultation are largely determined by how the planning and scheduling system combines the proposed exposures into the following hierarchy.

- Exposures → Alignments. An Alignment is a pointing of the spacecraft at a fixed position on the sky.
- Alignments → Obsets. An Obset is a collection of alignments done sequentially and which are sufficiently close together on the sky that the same pair of Guide Stars can be used.
- Obsets → Scheduling Units. A Scheduling Unit is a collection of obsets which must be done in a specific time sequence and is treated as a single unit by the scheduling software.

The model for the consumption of spacecraft time is that each obset will consist of:

- A slew from the previous obset's target position
- A guide star acquisition
- A series of alignments, each with a:
  - Exposure time
  - Data Readout time
  - Mechanism change time
  - A possible small angle maneuver time to get to the next alignment
- A number of occultation times and guide star reacquisition times which is a function of the total duration of all the alignments in the obset

Thus, if one knew how the planning and scheduling system combined exposures into alignments and obsets and knew the data readout and mechanism change times and small angle maneuver times, one could estimate the time spent in slewing or in occultation. With all this information, the total spacecraft time required by a scientist's proposal could be estimated.

Fortunately a system (called the *Transformation system* [3]) was being developed when this Resource Usage tool was being designed which:

- ordered the proposer's exposures into the Alignment-Obset-Scheduling unit hierarchy,
- determined the data readout and mechanism change times

This Transformation system is used to feed the HST Planning and Scheduling System.

Thus it seemed possible to use the Transformation system as the frontend for a Resource Usage calculator. The Resource Usage calculator would take the results of the Transformation system and then estimate the slew times, guide star acquisition times, small angle maneuver times, earth occultation times, and guide star reacquisition times. These times would be combined with the individual alignment times determined by the Transformation system to estimate the total spacecraft time required to execute the proposer's exposures.

Since the Transformation system also computes the data volume generated by the exposures and has the proposer's description of the observations in its database, the combined Transformation and Resource Usage systems could estimate the following constrained spacecraft resources:

- Total spacecraft time
- Data volume
- Spacecraft time which must be spent on the dark side of the earth
- Parallel observation time used
- Realtime uplinks required

as well as other quantities which the committee wishes to monitor.

By using the Transformation system as a frontend, the Resource Usage calculator would always use the same assumptions for how exposures were combined into alignments, obsets, and scheduling units; for the data readout times and mechanism change times; and for the data volumes as did the Planning and Scheduling system. For a single proposal considered in isolation, there would be no better way of estimating spacecraft resources.

4.1 Reusing The Transformation Rulebase

This Transformation system is written in OPS5, a language which the author was familiar after building the Duplication system. The program flow is controlled by a series of goals with the MEA strategy which gives extra importance to the first clause in the rule, usually the goal clause. The data structures, external declarations, rules for goal chaining, and the rules for each goal are all contained in separate small files.

The modular nature of the Transformation system made it apparent that it would be easy to graft a set of Resource Usage rules onto the existing Transformation system. Only two changes had to made to the internals of the Transformation system to use it as a frontend for the Resource Usage tool. The file containing the rules which chained from one goal to another was divided into two files, one containing the goals used by both Transformation and Resource Usage and one containing the goals used by Transformation alone in generating its output data files. The file containing the common goals would be combined with the goal chain used by Resource Usage to control the flow of the Resource Usage calculation. An additional data structure was added, with the necessary rule changes, to do a separate accounting of parallel exposures, i.e., exposures performed with a different instrument at

the same time as the primary exposure. Previously there had been no need to separately track parallel exposures.

These Transformation files would be compiled with the separate set of Resource Usage files to create the Resource Usage executable. These Resource Usage files contained the:

- Data structures.
- Goal chaining rules.
- External function declarations.
- Rules creating all the constants and lookup tables used.
- Rules which change data types to meet the assumptions of later rules.
- Rules summing exposure level resources into alignment level resources (mainly the data volume) and determining which exposures have narrow scheduling windows.
- Rules finding alignment level resource quantities, mainly the small angle maneuver times between alignments.
- Rules summing alignment level resources into obset level resources.
- Rules finding obset level quantities, mainly the occultation time and total spacecraft time.
- Rules summing obset level resources into proposal level resources.
- Rules finding maximum, minimum, and average values of proposal level resources for the different possible combinations of obsets.
- Rules reformatting the resource quantities into output formats.
- Rules writing the resources to the database and any applicable warnings to files.

Unlike the Duplication system, but yet like the Transformation system, Resource Usage made extensive use of external function calls to read from and write to the database, change datatypes, and perform mathematical calculations.

When the Transformation system was designed, using it to generate resource usage information was not a consideration of the design. This reuse of the Transformation system is mainly the result of the non-procedural nature of an OPS5 program, of the modular structure of the OPS5 files, and of the similarilarity of the data structures needed.

4.2 Isolated Modification of the Transformation Rulebase

With more familiarity with the Transformation rules, it became apparent that the different purpose of Transformation was embedded in some of the rules used both by Transformation and Resource Usage. The Transformation system was designed to operate on only exposures from one year in a proposal which may contain exposures for multiple years. The Resource Usage system was designed to estimate the resources for all years' exposures in a

proposal. Thus for Resource Usage to use the Transformation rulebase, it had to prevent Transformation from combining exposures from different years into an alignment or obset.

Preventing cross-year exposure combinations was accomplished by adding to one of the Resource Usage files two rules which in their first clauses referred to a goal which is in the part of Transformation rules used by both systems.

```
(p remove-different-cycle-exposure-links
    (goal
         ^has-name                    merge-exposures
         ^has-status                  active
         ^task-list                   find-potential-exposure-merges)
   {<mergeable-exposure-link>
    (mergeable-exposures
         ^first-exposure-number       <second-exposure>
         ^second-exposure-number      <first-exposure> ) }
    (exposure-properties
         ^has-exposure-number         <second-exposure>
         ^has-scheduling-cycle-name   <cycle-name> )
    (exposure-properties
         ^has-exposure-number         <first-exposure>
         ^has-scheduling-cycle-name   <> <cycle-name> )
-->
    (remove <mergeable-exposure-link>))

(p remove-different-cycle-alignment-links
    (goal
         ^has-name                    merge-alignments
         ^has-status                  active
         ^task-list                   find-potential-alignment-merges)
   {<link-to-remove>
    (mergeable-alignments
         ^has-first-alignment-order   <first-alignment-order>
         ^has-second-alignment-order  <second-alignment-order>
    (assignment-record
         ^has-Pepsi-exposure-number   <first-exposure>
         ^has-alignment-order         <first-alignment-order>)
    (assignment-record
         ^has-Pepsi-exposure-number   <second-exposure>
         ^has-alignment-order         { <second-alignment-order> <>
                                            <first-alignment-order> } )
    (exposure-properties
         ^has-exposure-number         <second-exposure>
         ^has-scheduling-cycle-name   <cycle-name> )
    (exposure-properties
         ^has-exposure-number         <first-exposure>
         ^has-scheduling-cycle-name   <> <cycle-name> )
-->
```

```
(remove <link-to-remove>) )
```

Since these rules are in a file which is not compiled when creating the Transformation executable, these rules have no effect on the Transformation system's operation. Because these rules are active during a Transformation portion of the goal chain, the rules are modifying the operation of the Transformation frontend to the Resource Usage system.

The non-procedural nature of OPS5 allows one user of a set of rules to alter the set's operation without affecting other users of that set of rules.

4.3 Resource Usage Status

The current Resource Usage tool uses 149 rules unique to it and 327 rules from the Transformation system. Resource Usage and Transformation are both limited at present in the number of exposures they can process. For proposals with more than about 800 exposures, the virtual memory required exceeds the 90,000 page virtual memory limit of the account used. However, these 800 exposures contain more than 6000 separate data elements for the OPS5 rules to match on - thus the large memory usage. For the current pool of GTO proposals, about 10% of the proposals have more than 800 exposures.

The speed of the Resource Usage tool is more important than for the Duplication tool. A proposal may be accepted with the condition that it be reduced to meet certain resource limits. This paring down and verifying that the resource limits are met will probably be an iterative process. Thus the Resource Usage tool may be run several times on a given proposal. It presently requires about two minutes of CPU time to run the Resource Usage tool on the average proposal and about thirty minutes of CPU time for the largest proposals which do not exceed the memory limits. These execution times are acceptable in verifying that proposals are meeting the set resource limits.

The large memory usage appears to be the result of very large numbers of partial instantiations of rules, i.e., the first few clauses in rules are satisfied by many data. The memory usage can be reduced by reordering both the clauses and the order of the elements within the clauses. Reordering some of the clauses has already reduced the memory usage by a factor of two, but this is not sufficient to process the largest proposals. The permanent solution to the memory usage problem will probably involve identifying disjoint subsets of the exposures, and operating on one subset at a time.

5 Strengths and Weaknesses of OPS5

The greatest strength of OPS5 is in its simple syntax, which allows a domain expert to learn quickly how to write rules in OPS5. The concept of data driven computation and the use of the MEA strategy with goal chaining provides the expert with a simple method of controlling the execution of the OPS5 program. With initial help and guidance from a person familiar with OPS5, domain experts can create their own expert systems.

As the expert system is created by the domain expert, rather than the Knowledge Engineer, the domain knowledge goes directly from the expert to the rules without going through the

"filter" of the Knowledge Engineer. This should decrease the chance of a misunderstanding being coded into the expert system. When the domain expert creates *AND DEBUGS* the expert system, there is a much greater chance that the system will do what the expert thinks it should.

Reducing the reliance on the Knowledge Engineer also lessens the bottleneck caused by the small number of Knowledge Engineer relative to the large number of "Experts".

Another advantage of OPS5 is the ability to reuse and modify existing OPS5 rules without affecting the other users of those rules. Due to the non-procedural nature of an OPS5 program, functionality can be added or changed by simply incorporating additional rules at compile time with any editing of the original code. This requires only that the rules are kept in small modular files. A core expert system can be written and then modified to suit the needs of other users. With a procedural language, making isolated modifications to common code is much more difficult. To quote the Computer Scientist who first introduced the author to OPS5, "Who says you can't write reusable code?"

The biggest disadvantage of OPS5 is the environment. One cannot examine the text of the rules while single stepping through the rules with the command interpreter. Thus for debugging, either a paper copy of the rulebase is needed or a multi-window hardware system is needed. An environment where one can view the rulebase, change rules, recompile, relink, and run the command interpreter would be a great aid to developing OPS5 expert systems.

The other major disadvantage of OPS5 is the large memory usage. There are ways to reduce the memory requirements by changing the code, and the availability of computer memory is always increasing. However, large memory usage is by its nature data dependent and does not constitute a fundamental limitation of the language.

References

[1] *VAX OPS5 User's Guide*, 1985, Digital Equipment Company

[2] *VAX OPS5 Reference Manual*, 1985, Digital Equipment Company

[3] Rosenthal, D., Monger, P., Miller, G., Johnston, M. 1986, *Proceedings of the 1986 Conference on Artificial Intelligence Applications*, NASA Goddard Space Flight Center

ROBOTICS