JPL

WORKSHOP NOTES

International Workshop on Planning and Scheduling for Space Exploration and Science

Distribution limited to workshop participants

Program Committee

Steve Chien, Jet Propulsion Laboratory (Chair)
Nicola Muscettola, NASA Ames Research Center (Co-chair)

Chester Borden, Jet Propulsion Laboratory
John Bresina, NASA Ames Research Center
Robert Connerton, Goddard Space Flight Center
Jon Erickson, Johnson Space Center
John Jaap, Marshall Space Flight Center
Glenn Miller, Space Telescope Science Institute
Ben Smith, Jet Propulsion Laboratory
Stephen Smith, Carnegie-Mellon University

28-30 October 1997
Oxnard, CA, USA

Preface

In recent years, automated planning and scheduling technology have matured - resulting in a number of successful, deployed systems. Within the area of Space Exploration and Science, planning and scheduling is directly relevant in a wide range of areas, including:

- spacecraft and payload operations; operations of space-borne and ground-based scientific observatories;
- load forecasting, resource allocation, and scheduling of antenna resources;
- generation of antenna tracking plans;
- science data analysis; analysis of operability of designs; and
- operations and payload scheduling for space transportation systems.

These breadth and importance of these areas represents both an unprecedented opportunity to impact space exploration and science and a unique set of challenges for planning and scheduling technology. On the eve of this workshop, we are particularly delighted to have significant representation (over 60 participants) drawn from three diverse groups relevant to developing and deploying planning and scheduling technology for space exploration and science:

- NASA, other space agency, Industry, and University technologists working on planning and scheduling applications for space
- User group representatives who have problems and requirements, and
- Artificial Intelligence and Operations Research technologists working in the area of automated planning and scheduling with technologies relevant to space exploration and science.

This workshop is the first of a new regular forum to foster exchange of information, collaborations, and teaming to develop and deploy planning and scheduling technology within NASA and the space community. The workshops will be held every twelve to eighteen months - the next workshop will be held in northern California, and will be sponsored/hosted by the NASA Ames Research Center.

The goals of the workshop are to:

- Inform user groups within NASA and the space community about technologies available and being developed and their levels of maturity.
- Inform NASA and other space AI practitioners about user groups, their problems and requirements.
- Inform researchers in academia about space application domains.
- Encourage contacts between technologists and users.

This workshop would not have been possible without the tireless effort and dedication of numerous individuals and organizations. I would like to thank my co-chair, Nicola Muscettola and the rest of the program committee: Chester (Chet) Borden, John Bresina, Robert (Bob) Connerton, Jon Erickson, John Jaap, Glenn Miller, Ben Smith, and Stephen Smith. Their efforts in reviewing the plethora of submissions and eliciting participation enabled the diverse and high quality program that we have for this workshop. I would also like to thank Melvin (Mel) Montemerlo, of NASA Headquarters, whose support of and commitment to Artificial Intelligence Technology over more than 10 years is a primary factor in unprecedented opportunities today in planning and scheduling for space. Finally, this workshop would not have been possible without the efforts of the Pat McLane, Helga Mycroft, and Shannon Connolly of JPL conference administration services.

Steve Chien

Pasadena, CA
October 1997

Table of Contents

A Post-Process Optimization Algorithm for Resource Constrained Project Scheduling

Alex Aldas

The Boeing Company
13100 Space Center Blvd.
Houston, Texas 77059
Alex.Aldas@boeing.com

Abstract

As modern engineering projects grow more complex in nature, proper usage of available resources has become increasingly important. Inefficiencies in a scheduling engine could leave a project overdue and over budget. As a result, schedule density and schedule makespan have been drawn to the forefront of the planning and scheduling community. This paper does not try to solve the overall scheduling problem of maximizing resource usage, but rather explores the use of a post-process optimization scheme that attempts to shorten project makespan and increase resource usage through the use of slack distribution.

Introduction

In recent years the National Aeronautics and Space Administration (NASA) has undergone a culture change within its operations to one of "faster, better, cheaper." This philosophy has propagated into every facet of NASA, but arguably none so much as the planning and scheduling community.

The production of large, complex, one-of-a-kind products is costly in-of-itself. Unnecessarily long schedules that do not fully utilize available resources can have a dramatic effect on the overall cost of a given project. A reduction in project makespan by as little as one day has a potential cost savings in the tens to hundreds of thousands of dollars. Therefore, the goal of the scheduler in assembly operations would be one of minimizing project makespan without lowering the quality of the schedule.

Mission planning and scheduling provides a different flavor of the resource constrained project scheduling problem (RCPSP). Here no product is assembled, nor is there a defined network of activity. Rather, mission planning involves the scheduling of individual or groups of activities, which may contain temporal and/or resource constraints. Effective scheduling would try to maximize the number of high priority activities that can be performed during the given mission time frame, while placing lower priority activities in the timeline gaps that remain.

Whether the problem involves spacecraft vehicle assembly or space station mission planning, scheduling has a direct impact on the cost and efficiency of the project. This paper does not attempt to explain the methods used to schedule such projects, but rather it presents a post-process *schedule packing*[1] algorithm which strives to further condense *any* feasible schedule. By packing a schedule, excess slack is redistributed and later extracted. The final schedule has a decreased cycle time and resource idle time, which corresponds to project cost savings.

Resource Constrained Project Scheduling

A resource constrained project scheduling problem is defined by a set of tasks, which operate on a set of finite capacity resources. Each task can contain any number and variety of constraints imposed. The goal of such a problem is to define a schedule in which all constraints are satisfied while meeting some objective function, usually to minimize project makespan. Any schedule which satisfies all the constraints is labeled as *feasible*, while any feasible schedule whose makespan is as short as the shortest known feasible schedule is labeled as *optimal*.

Schedule Packing

Schedule packing is a *post-process* optimization scheme meaning that it attempts to further optimize an existing feasible schedule with respect to assignment density. As such, the algorithm is independent of the method used to generate the feasible schedule. While it does not perform scheduling itself, it *drives* a scheduler to select a task, unschedule it, and re-schedule it on the timeline. Schedule packing can be performed on any feasible schedule so long as the scheduler has the capability of "locking" a schedule and selectively unscheduling and re-scheduling individual tasks. It is therefore the responsibility of the scheduler to insure that all constraints are satisfied in the re-scheduling process.

The easiest way to explain schedule packing is by example. For simplicity, let us take the most trivial of all

[1] The Boeing Company has applied for a patent on this algorithm.

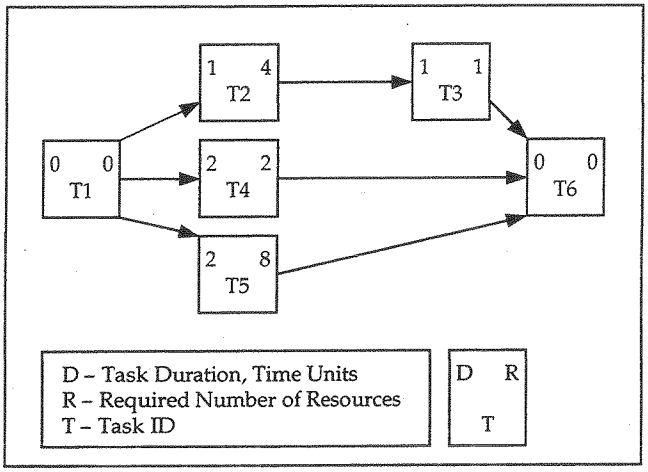


Figure 1: Example project in a precedence diagram.

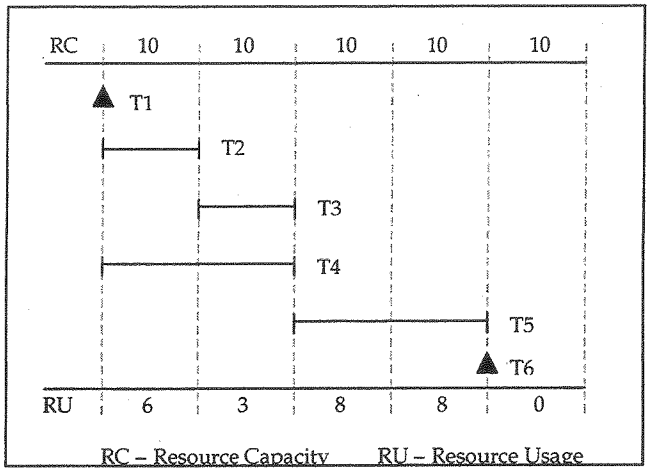


Figure 2: Initial feasible schedule

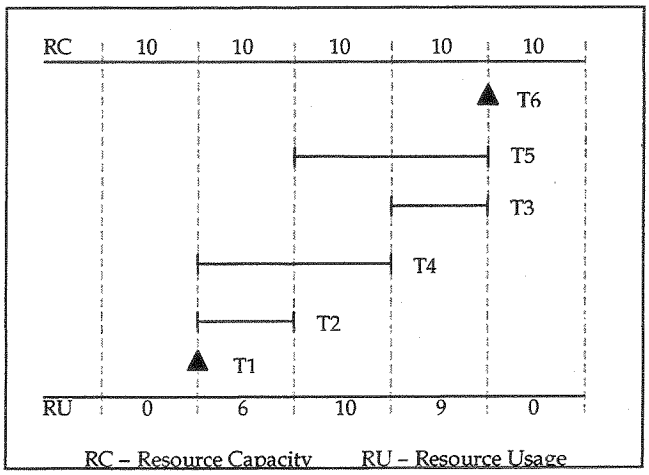


Figure 3: Right shifted schedule

cases: single resource, with constant initial availability over time. For this example, we will define a resource R with a constant capacity of 10. Our project, depicted in Figure 1, consists of 6 tasks with tasks T1 and T6 being milestones. Each task has associated with it a duration, in time units, a required number of resources R, and a task ID. Performing a sort by name, followed by a sort by predecessors on the task list and scheduling each task in the first available interval (left-most feasible interval) results in the feasible schedule illustrated in Figure 2.

Our initial schedule has a makespan of 4 time units. If we were to track our resources, i.e., not enforce resource constraints, we could achieve a schedule makespan of 2 time units. Hence, the added length to the project duration is due to resource constraints rather than technological (precedence) constraints. To shorten the makespan of this schedule we must be able to resolve critical resource constraints within the schedule.

Schedule packing begins with our initial feasible schedule. The algorithm proceeds as follows:

- Select all tasks.
- Sort by latest scheduled end time (secondary sort by latest start time in case of a tie).
- Starting from the top of the task list, unschedule each task and reschedule it into the right most feasible interval.

The results of the above process on our initial feasible schedule are illustrated in Figure 3. What we have done is utilize the slack that exists within our initial schedule to "pack" tasks up against the right hand fence, i.e., the project end time. Through inspection of Figure 3, the actual project duration has been shortened by 1 time unit. Shifting the entire project back to the left by 1 time unit results in a new feasible schedule with a makespan of 3 time units.

The next logical question is can this new schedule be packed further? A good indication of whether a schedule has the potential of being packed is to examine the slack associated with the beginning task(s) of a project. If any of the beginning tasks has a slack time greater than zero, then the project can potentially be packed. Our initial schedule in Figure 2 indicates that task T4, a beginning task since T1 is a milestone, has a slack time of 1 time unit. This indicates that our initial schedule has the potential of being packed. After performing schedule packing on our initial schedule, we find that all of the beginning tasks, T2, T3, and T5 have a slack time of zero. Since all of the beginning tasks have a slack time of zero, the resulting schedule has probably been packed as much as possible.

Schedule Packing with Real World Projects

Real world projects provide a much more complex problem than the example project described above. In the example problem we assumed a single resource with a constant capacity throughout all of time. In real world projects, however, such simplifications are rarely found. In modern

aircraft assembly, the scheduling of a single sub-assembly can have as many as 1,164 tasks scheduled against 226 resources. In addition, the availability of skilled labor is usually a function of shift and contract work.

Our scheme as described above is inadequate for packing complex projects. The primary reason for this short-coming is because we cannot simply pack the schedule against the right-hand fence and then slide the whole schedule back towards the left-hand fence. This would only be possible if all resources used within the project have a constant initial availability. Since most real world projects do not have this property, we must extend the schedule packing scheme to handle the generic case of variable resource availability.

By sorting and rescheduling against the right-hand fence we attempt to distribute slack among tasks so that resource conflicts can be broken. The result is a "packed" schedule crammed against the right-hand fence. If we perform the exact same operation but in the reverse direction, i.e., sort by earliest scheduled start time and schedule into the left most feasible interval, we are effectively "sifting" tasks into "slack holes" left by the shift right operation. Further "sifting" will result in a steady-state schedule which is packed in both directions. Further shifting will produce the same initial schedule.

To summarize the generalized schedule packing algorithm:

- Select all tasks.
- Sort by latest scheduled end time (secondary sort by latest start time in case of a tie).
- Unschedule and re-schedule each task into the right most feasible interval.
- Sort by earliest scheduled start time (secondary sort by earliest finish time in case of a tie).
- Unschedule and re-schedule each task into the left most feasible interval.

The generalized schedule packing scheme is independent of project structure, resource availability, and resource requirements. It can be used on any feasible schedule so long as the underlining scheduler is capable of unscheduling and rescheduling a single task into left-most and right-most feasible intervals.

Benefits of Schedule Packing

The degree to which schedule packing can reduce project makespan is a function of several variables including: 1) quality of the initial schedule and 2) variability in resource availability.

Initial schedules that are near optimal contain a limited amount of slack for which schedule packing can be utilized. Since slack is what the algorithm uses to pack tasks against each of the project fence dates, a schedule with a large amount of slack is more apt to reductions in schedule makespan than a highly optimized initial schedule.

	Makespan, hh:mm	*% Compression*
Benchmark Test 2		
SFI (Baseline)	685h:16m	-
SFI + SP	591h:54m	14.62
Benchmark Test 3		
SFI (Baseline)	861h:25m	-
SFI + SP	696h:25m	21.18
Benchmark Test 4		
SFI (Baseline)	845h:40m	-
SFI + SP	654h:8m	25.54

* Note: Simple Feasible Interval (SFI) scheduler used with a sort by predecessor dispatch order. SP = Schedule Packing

Table 1: Compression analysis for benchmark tests

Schedules that contain a high degree of variability in resource availability usually result in projects with numerous resource conflicts. While schedule packing can eliminate some of these conflicts, its performance is limited.

To examine the degree of compression that schedule packing provides, a set of benchmark tests[2] that emulate real-world assembly projects was chosen. The analysis consisted of tests 2, 3, and 4 within the 12 test data set. Each test contained 575 tasks to be scheduled against 17 resources using simple resource modeling – single mode with precedence, labor and zone constraints. Results from the benchmark tests are presented above in Table 1.

While this data set is a fair example of an assembly process, in practice we have found that the constraints modeled are much more complex than those found in the benchmark tests. Additionally, resource availability is not as "cleanly" defined. Modern assembly processes may move between two and three shift operations and may pull additional resources based upon contract needs. As a result, resource availability histograms appear more like skylines than simple step functions.

To test schedule packing in real world projects, an aircraft sub-assembly with a degree of constraint modeling and resource utilization typical of modern complex assembly operations was chosen. The "live" data set included 169 tasks to be scheduled against 42 resources. Resource requirements were modeled in a multi-modal form and included advanced modeling techniques such as interruptibility, grouping, linking, and timeline exclusivity. The results of the test are displayed in Table 2.

	Makespan, hh:mm	*% Compression*
Baseline Schedule	115h:45m	-
Baseline Schedule + SP	98h:9m	15.2

* Note: Schedule generated by TimePiece©. SP = Schedule Packing.

Table 2: Schedule packing results on aircraft sub-assembly scheduling.

[2] Benchmark tests can be found at http://www.neosoft.com/~benchmrx

From experience, schedule packing can compress a "good" initial schedule by 15-20%. The resulting schedule has a higher resource utilization, which usually corresponds to a decrease in schedule makespan. While schedule packing was described in the context of assembly operations, it has been successfully used within the planning and scheduling operations of facility resource management[3] and on-orbit operations planning[4] with similar results.

Conclusion

Schedule packing is a post-process optimization algorithm that drives a scheduling engine to compress any existing feasible schedule so long as the scheduling engine is capable of enforcing all task constraints. It has been shown in practice to reduce the schedule makespan of "good" initial schedules by 15-20%. In a time where proper resource management is of high priority, schedule packing provides a simple schedule improvement scheme that many of today's schedulers can employ to obtain a better resource utilization rating.

References

Wiest, J. D. 1964. Some Properties of Schedules for Large Projects with Limited Resources. *Operations Research.* 12(3):395-418.

---

[3] F-18 Avionics Integration Lab, The Boeing Company (FRMS©)

[4] SpaceHab Operations (COMPASS©)

Scheduling Research and Applications

Mark Boddy
Mark Ringer

Honeywell Technology Center

Overview

In the Scheduling Group at the Honeywell Technology Center (HTC), we are engaged in a wide variety of research and development projects related to scheduling for space and aviation. Most of these projects result in implemented systems, several of which have become, or will soon become, fielded applications, some of them solving problems substantially more complex and/or orders of magnitude larger than appear in the current applications literature. Our current research efforts are extending the utility of scheduling systems for complex dynamical models, for distributed applications, and for mixed-initiative systems.

For space applications, we have developed scheduling systems for Space Station experiment scheduling, information processing scheduling for the Earth Observing System (EOS), and ground support for satellite operations. In the aviation domain, we have completed or are currently pursuing applications in aircraft avionics, airport ground operations scheduling, and airline maintenance scheduling. Four of these applications are briefly described below.

Approach

Our technical approach to scheduling, along with most of our implemented systems, is based on Constraint Envelope Scheduling (CES). CES is a form of "least commitment" scheduling, in which schedule construction and modification is accomplished through the incremental addition and deletion of constraints on activities' temporal relations and resource usage. The result is a schedule in which each activity is only constrained to execute within a certain time window. Contrast this with conventional timeline approaches that place each "activity" at an exact location on a timeline. The CES approach allows for more flexible search, a focused analysis of bottlenecks or conflicts, and easier dynamic modifications to an existing schedule.

The CES approach was developed specifically to address some of the issues that arise in implementing scheduling systems for large organizational applications, for example manufacturing or spacecraft operations. The issues we were concerned with include large-scale problems (hundreds to tens of thousands of activities), mixed-initiative schedule generation, incremental, iterative rescheduling during schedule generation and execution, complex domain models, and the need to explain to a human user the reason for a scheduling conflict or infeasibility. This approach has now been demonstrated to provide benefits for those problems, across a wide range of application domains, with solution methods ranging from (almost) fully-automated search to manual scheduling with constraint-checking and culprit identification functions.

In our recent research, we have discovered additional benefits to CES, initially unlooked-for but turning out to be very promising. First, it turns out that modeling the scheduling process explicitly as the accumulation of additional constraints supports a flexible and natural semantics for distributed scheduling. This advantage holds for several interpretations of the term "distributed." In an application where multiple administrative or bureaucratic entities are modifying a common schedule (think Shuttle mission planning), the ability to tag added constraints with information regarding who added them and why permits us to support a principled negotiation between competing interests in the construction of a common schedule. For a more strongly-distributed application such as Air Traffic Control or supply chain management, explicit constraints provide a semantic basis for negotiated commitments among entities who may be unable or unwilling to share complete information regarding their local schedules or objective functions. For either of these cases, retaining flexibility in both schedule generation and execution (the "envelope" in CES) simplifies the negotiation process through the avoidance of premature commitment.

An additional strength of the CES approach has resulted from our explicit separation of the discrete and continuous aspects of scheduling problems. This "hybrid reasoning" approach has been accomplished through the implementation of a continuous-domain temporal reasoning engine with a very specific set of properties, including efficiency (millisecond incremental updates for constraint addition *and* deletion), completeness and correctness with respect to consistency, caching of partial information for incomplete constraint propagation back into the discrete model, and "culprit identification:" the ability to trace an inconsistency or infeasibility back to the set of interacting requirements or scheduling decisions that were responsible. Almost without recognizing what we were doing, we have ended up implementing a novel architecture for the principled solution of hybrid (mixed discrete and continuous) constraint problems, borrowing techniques from both AI and Operations research.

In current work, we are extending this architecture for the solution of problems with a more complex continuous model, for example refinery scheduling, which requires at least a general LP model, and possibly an NLP. Our preliminary results indicate that the LP integration, at least, is quite feasible, and that the resulting system will be able to solve complex scheduling and optimization problems at least a couple of orders of magnitude larger than the best "Mixed-Integer Linear Programming" models currently available. Some domains for which this increased expressive capability may be useful include manufacturing, aircraft mission planning and flight dynamics, and spacecraft or satellite operations explicitly incorporating orbital dynamics.

Application descriptions:

Satellite ground ops

HTC has developed a prototype satellite ground operations scheduling system. The targeted domain is a network of space-based satellites in various orbits with a network of ground-based earth stations. Each ground station and satellite has a set of both unique and shared resources. Activities are scheduled to use both satellite and ground stations based upon criteria of availability, ephemeris data, and resource constraints. The satellites orbit in various orbits (from LEO to GEO) imposing constraints on activities by their visibility windows. Ground stations contain the communication antennae. The ground-based equipment can either assigned to a ground station or shared across a network of ground stations. Activities have start and end times as well as a variable length duration. Each activity requests a set of possible ground stations as well as a set of equipment that it needs. The type of satellite necessary is also specified.

Distributed image data archiving and analysis

Under contract to the NASA Goddard Space Flight Center's (GSFC) Information Science and Technology Branch, HTC has developed a scheduling system for the Earth Observing System Data and Information System (EOSDIS) Distributed Active Archive Center (DAAC). One of the challenges is to ensure quick and easy retrieval of any data archived within the Distributed Active Archive Center Data Archive and Distributed System (DAAC DADS). Accessing that amount of information is a formidable task that will require innovative approaches. In order to improve the performance of the DADS, to make better use of limited resources, prevent backlog of data, and to provide information about resource bottlenecks and performance characteristics, a constraint-based task and resource scheduler was developed by the Honeywell Technology Center. This scheduler is in the process of being fielded by contractors for NASA Goddard.

Airline maintenance

HTC is developing an airline line maintenance scheduler for integration into an existing Honeywell product. The scheduler allows the human maintenance scheduler the ability to automatically generate a schedule, check consistency on manual changes, and a more useful graphical window into the operation of the airline. We have multiple activity types to model recurring checks and activities with deadlines based upon flight hours or takeoff/landing cycles. Multiple types of resources are modeled: labor (with work calendars), tools, labor, and maintenance stations (hangars, etc.). The schedule can also be generated with a mixed-initiative paradigm, the human being responsible for some of the heuristic information, while the scheduler is responsible for the rest of the heuristic information and the laborious activity of enforcing consistency between all of the activities. The user can generate reports such as

a list of the activities to be accomplished on each aircraft or for each employee.

Aircraft avionics

Honeywell is now delivering the new Boeing 777 integrated Airplane Information Management System (AIMS). AIMS functions range from hard real-time to non-real-time, and include flight critical and non-essential functions, and must all reside on the same distributed multiprocessor system. This system, based on ARINC 659[1], allows for high resource utilization while providing strict partitioning and provable performance via pre-scheduling of processing and communication resources. This application was large, complex, and hard to solve. To be a little more concrete: "large" means almost 30,000 activities, "complex" means several activity types, periodic behavior, and assorted types of temporal constraints, and "hard to solve" means that we have been unable to eliminate backtracking through the use of search heuristics. The schedule generated by HTC is currently running on the 777 providing critical timing information for all AIMS processing tasks and communications messages.

Planning and Scheduling User Services for NASA's Deep Space Network

C. Borden, Y.-F. Wang, G. Fox

Background

NASA is currently developing and operating "faster/better/cheaper" unmanned space flight missions. This is resulting in a significant increase in the number of missions NASA's Deep Space Network (DSN) will have to support. Increases in the number of missions to be supported is occurring concurrently with these new missions (1) testing new spacecraft telecommunication technologies, (2) having less project resources available for operations planning related to the DSN, (3) a shorter development cycle from project start to spacecraft launch, and (4) rapid, realtime changes in mission coverage requirements (e.g., Mars Pathfinder). In this environment, DSN management faces an increased demand for decision-making on a larger number of more uncertain missions, and with quick response time.

New ground telecommunication system technologies and operating procedures are also being developed by the DSN to provide more effective mission support. New technologies and approaches include (for example):

- simultaneous communication with multiple spacecraft from a single ground antenna
- spacecraft using multiple, arrayed ground antennas
- automated, self-configuring, unmanned ground stations
- automatically processing high level DSN service requests into detailed, individual equipment level requirements
- automated DSN Complex and/or Station operations
- reduced antenna pre- and post-calibration (setup) times.

The rapid change in mission and ground system capabilities implies a significant impact on DSN's ability to optimize their capacity evolution plans and mission scheduling on ground resources. In addition, rapid change in information technology and desktop computing capability provides an opportunity to use modern software architectures and tools for improved decision-making. Analysis tools can now readily interoperate with relational databases, Web browsers and other value added software tools and environments. Optimization, simulation and computational intelligence techniques continue to advance in terms of desktop capability and software interoperability.

To adequately capture the effect of these changes in the DSN and space flight project environment, JPL is implementing a prototype analysis capability that provides data access, load forecasting and scheduling in a single integrated environment. This system links to a relational database and shares the same user interface for both DSN and space project users. Existing analysis tools provide a useful point of departure for improving the DSN's planning and scheduling capability.

The Current System

JPL currently uses the FASTER Tool Suite for DSN load forecasting and scheduling (Loyola, 1993), (Fox and Borden, 1994). This capability has been used for many years as the basis for DSN commitments of ground resources to missions.

Major FASTER capabilities include:

- Load forecasting
- Schedule editing
- Viewperiod graphical display and management
- Antenna usage listing (including unused time)
- Graphical display of critical events over time

The FASTER software was designed to answer questions relating narrowly to DSN load forecasting and mission scheduling. With the change in NASA direction to a large number of small missions, life-cycle cost-driven mission commitments, and a new programmatic focus on Mars using multiple simultaneous spacecraft, operating the current FASTER software system is becoming very time-consuming. Multiple local data sets and non-integrated and unoptimized tools contribute to an inefficient user environment. Though FASTER is useful for DSN planning and scheduling, the opportunity to improve these tools (and the embedded processes) can lead to significant improvements in data management, timeliness of analyses, and extended analysis capability to evaluate and optimize new technologies and approaches.

Proposed Implementation

A new prototype tool is being implemented in response to requirements for extended load forecasting and scheduling capability and to provide mission and DSN users with an integrated, graphical environment for mission design and ground system planning and scheduling studies. The architecture implementing this integrated system, TIGRAS (Covington and Wang, 1997), provides both DSN and space flight project users access to the same computational engine and database (MADB) (Zendejas, 1997). The TIGRAS prototype provides an end-to-end analysis environment that includes an embedded work flow manager, secure access to user-owned products, configuration management, and an audit trail.

The TIGRAS user model is designed to improve efficiency in access to data and tools, establish use of DSN ground support as a mission design variable, and facilitate better

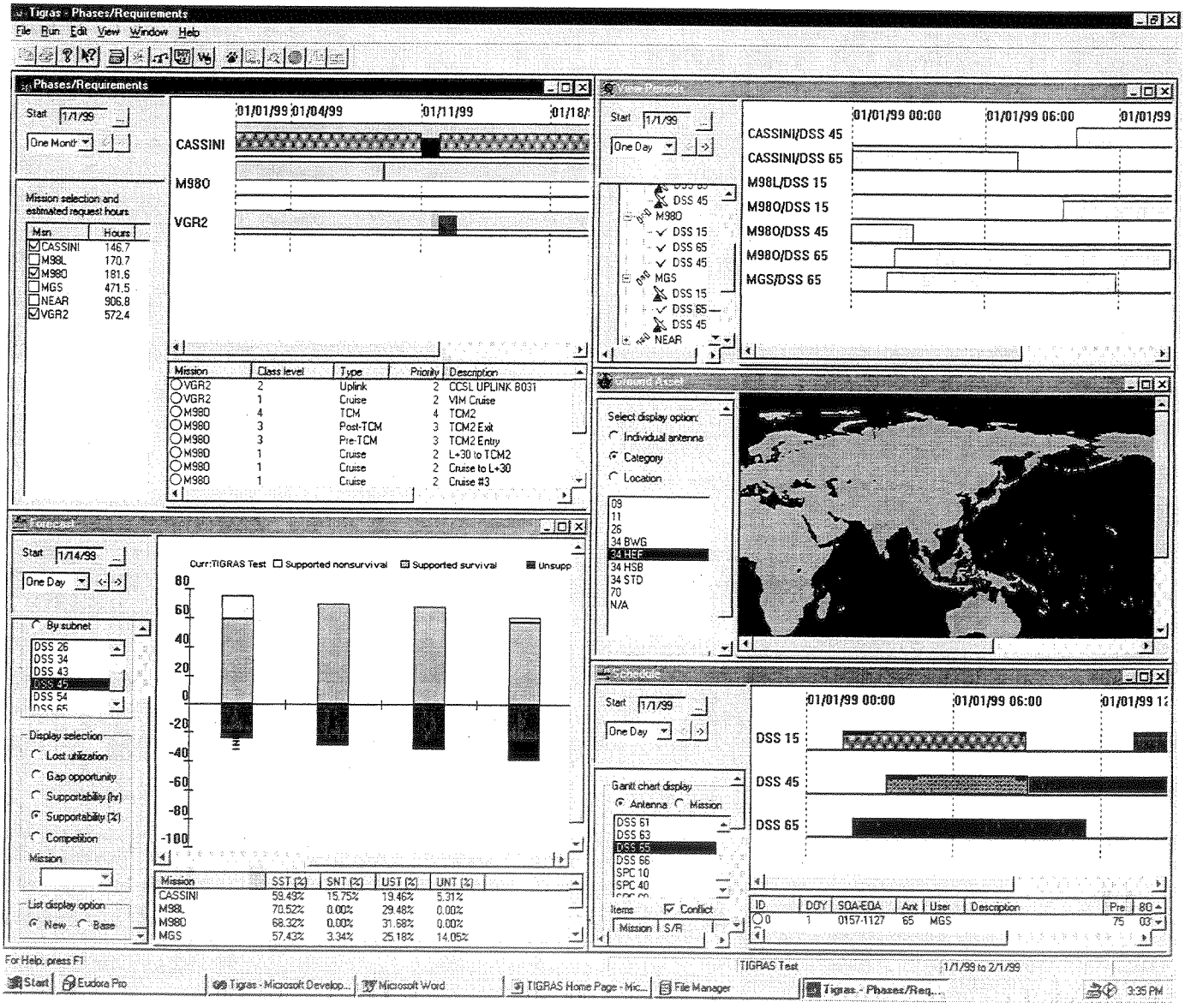
communication between missions and DSN. Mission users can access quantitative baseline DSN loading and scheduling products and tools for further analysis. Links to trajectory products and visualization are supported through MADB. Web-access to TIGRAS data is available. Software to support online schedule change requests with rapid conflict resolution is being prototyped. The anticipated new approach for user interface with the DSN called the Service Request Processor will also be supported.

TIGRAS is a Windows-based application with linkage to MADB (SQL server) via ODBC (Open Database Connectivity). Figure 1 is the screen of the TIGRAS environment. Users can examine requirements, view periods and antenna properties then perform forecasting and scheduling studies all in the same environment, using the standard Windows operating system.

Future Developments

In future versions computational intelligence approaches such as Neural Networks (Wang, Cruz and Mulligan, 1994) and Fuzzy Logic will be used to enhance the current forecasting algorithms to provide an extended, more optimized, automatic allocation engine. This engine expands the scheduling scope from antenna level to equipment level allocations with more constraint checking and optimization. Using the current scheme, what-if simulation and sensitivity analysis can be preformed to optimize the allocation. On the software side, we are investigating DCOM and MS Transaction server approaches to enable TIGRAS functionality as reusable, distributed objects. This will ensure an extensible, maintainable object implementation of functions used in this enhanced resource allocation analysis process.

FIGURE 1:



References

S. J. Loyola, 1993. "PC4CAST—A Tool for DSN Load Forecasting and Capacity Planning," *The Telecommunications and Data Acquisition Progress Report 42-114, vol. April-June 1993*, Jet Propulsion Laboratory, Pasadena, California, pp. 170-184.

G. Fox and C. Borden, 1994. "Low-Earth-Orbiter Resource Allocation and Capacity Planning for the DSN Using LEO4CAST," *The Telecommunications and Data Acquisition Progress Report 42-118, vol. April-June 1994*, Jet Propulsion Laboratory, Pasadena, California, pp. 169-178.

R. Covington and Y.-F. Wang, 1997. "TMOD Integrated Ground Resource Allocation System (TIGRAS) Functional Requirements Document," *draft*.

S. Zendejas, 1997. "MADB Baseline Design Working Document," *draft*.

Y.-F. Wang, Jose B. Cruz, Jr., J. H. Mulligan, Jr., 1994. "Adaptive Scheduling Utilizing a Neural Network Structure," *PROCEEDINGS ICNN 94*, VOL 6. pp.3806-3811.

Optimizing Observation Scheduling Objectives

John L. Bresina
Recom Technologies

Robert A. Morris
Florida Institute of Technology

William R. Edgington
Recom Technologies

NASA Ames Research Center, Mail Stop: 269-2
Moffett Field, CA 94035-1000 USA

{bresina, morris, wedgingt}@ptolemy.arc.nasa.gov

Abstract

In this paper, we present a novel approach that enables the automatic generation of high quality schedules with respect to a given objective function. The approach involves the combination of two techniques: GENH, which automatically generates a search heuristic specialized to the given problem instance, and HBSS, which employs the heuristic as a bias within a stochastic sampling method. We empirically investigate the performance of these techniques, individually and in combination, within a real-world application of observation scheduling.

Introduction

In this paper, we present a novel approach that enables the automatic generation of high quality schedules with respect to a given objective function. The approach involves a combination of two techniques: GENH [7], which automatically generates a search heuristic specialized to the given problem instance, and *Heuristic-Biased Stochastic Sampling* (HBSS) [2], which employs a given heuristic as a bias within a stochastic sampling method. These approaches have been implemented within the Associate Principal Astronomer (APA) system, which provides management support and improved observation scheduling for fully automatic, terrestrial telescopes.

Deriving search heuristics that are both accurate and computationally inexpensive is a difficult endeavor for most problems. This is especially true when not just any solution is acceptable and the heuristic is further required to find a high quality solution. Furthermore, the larger the class of problem instances, the more difficult it is for a search heuristic to perform consistently well over the class.

GENH was initially motivated by the idea that the role of a domain expert should be limited to constructing the domain-specific objective function. Before GENH, the search heuristic had to be manually tuned using a "generate and test" exploration, which was a time-consuming and difficult process that no one wanted to repeat very often. However, within this application domain, the nature of the scheduling problem changes day-to-day due to the changing position of the celestial targets as well as the addition or modification of the observation requests. Hence, the search heuristic that performs well for one day's scheduling problem may not do so well for the next one. Since it is difficult to find a single heuristic that performs well across the wide range of problem instances, the automation of this task by GENH is important for this scheduling application.

The number of feasible schedules of telescope observations for a given day is on the order of 10^{100}; given such large search spaces, it is very unlikely that any local heuristic can yield a globally optimal schedule via greedy search. The underlying assumption behind the HBSS approach is that strictly adhering to a search heuristic often does not yield the best solution and, therefore, that exploration off the heuristic path can prove fruitful. Within HBSS, the balance between heuristic adherence and exploration is controllable by the user. The accuracy of the search heuristic is an important factor in choosing the appropriate balance to use; typically, the less accurate the heuristic, the weaker the bias towards heuristic adherence should be. Another important factor is the amount of solution generation time available; if there is not much time available, then exploration must be limited and a stronger bias is recommended.

The outline of the rest of the paper is as follows. We first briefly present background on the application domain, and we next present the GENH and HBSS techniques. Then we empirically demonstrate the improvement yielded by the two techniques, individually and in combination. The improvement is with respect to the original dispatch scheduling technique encoded in the telescope controller. In the last section, we offer some concluding remarks.

Observation Scheduling Domain

The input to the APA observation scheduler is a set of requests, expressed using the Automatic Telescope Instruction Set, or ATIS[1]. Each request is composed of a sequence of telescope movements and instrument commands, as well as scheduling constraints and preferences.

A request is *enabled* on a given night if all of its constraints are satisfied. The primary constraint is that each request can be executed only within a specific time

interval. On a given night, an observation can only be executed during the intersection of the observation's interval and that night's interval of darkness. Enablement is also affected by the moon – each observation includes a constraint regarding whether the moon must be up, down, or either. Furthermore, even those observations that are enabled when the moon is up cannot be executed when its observation targets are too close to the moon. We refer to the time interval (for a given night) during which an observation can begin execution as its *enablement interval.* The primary preference specified is the relative priority that an astronomer assigns to each observation request. On many nights, not all of the possible requests can be executed; these relative priorities help determine which subset to execute on a given night. Each observation typically takes around five minutes to execute, and between 60 and 140 observations can be executed during a night.

We formulate this observation scheduling problem as a *refinement search* (also called *constructive search*). The scheduler's search space is organized chronologically as a tree, where the root node consists of the world model state at the beginning of the night. Each arc out of a search tree node represents an enabled request. The task of the APA scheduler is to find a sequence of observations that achieves a good score according to the objective function. For further domain details, see [3; 5].

The ATIS standard also specifies an heuristic dispatch policy which can be used to select the next observation to execute. The policy is expressed as four selection rules: *priority*, *number-of-observations*, *nearest-to-end-window*, and *file-position*. When the controller has finished executing an observation, it first determines the currently enabled observations and then applies these four rules in the sequence given to select one. Each rule is used to break ties that remain from the application of those that preceded it. If the result of applying any rule is that there is only one observation remaining, that observation is selected for execution and no further rules are applied. Since there can be no file-position ties, the dispatch policy is deterministic.

ATIS dispatch is a robust scheduling method that has been used fairly successfully for several years to schedule automatic photoelectric telescopes at Fairborn Observatory before the development of the APA. (See [6] for a performance evaluation of ATIS dispatch.) The dispatch decisions are determined purely locally, without look-ahead; by contrast, the APA uses a search-based scheduler (for APA scheduler details, see [4]).

The GenH Technique

In this section, we describe the GENH technique, first introduced in [7]. GENH automatically generates a search heuristic that is specialized for the given problem instance. For observation scheduling, the input to GENH consists of an objective function, a set of candidate heuristic attributes, and the set of observation requests. The output from GENH is the best multi-attribute heuristic found as the result of a search through

```
GenH(objective, attributes, requests)
begin
{generate seed}
bestH = random_seed(attributes)
best_score = eval(bestH)
{initialize boundary variables}
boundary = 0.5
granularity = 0.1
{repeat until granularity minimum exceeded}
repeat until granularity < 0.004
    {tune each attribute separately}
    for each A ∈ random_order(attributes)
        candidates = neighbors(bestH, A,
                               granularity, boundary)
        if (∃ H ∈ candidates such that
            eval(H) < best_score)
        then update best_H and best_score
    endfor
    {reset boundary variables}
    boundary = boundary / 5.0
    granularity = granularity / 5.0
endrepeat
return bestH
end
```

Figure 1: The version of the GENH heuristic generation algorithm employed in the reported experiments. The function `eval` invokes the scheduler and applies the objective function to the resulting greedy solution.

a space of candidate heuristics. GENH solves an optimization problem, defined as follows. The candidate solution set, $\mathcal{S}$, is $\{< w_i > \mid \forall_i w_i \in [0,1]$ and $\Sigma_i w_i = 1\}$; *i.e.*, the set of all weight vectors such that each weight is in the interval $[0,1]$ and the sum of the weights equals 1. GENH's evaluation of a candidate heuristic is the objective function score of the schedule found *via* greedy search using that candidate.

GENH uses *local search* (also called *repair search*) to conduct a search through $\mathcal{S}$. In a local search space, each node corresponds to a solution candidate, and for each node, a *neighborhood function* defines a set of neighbors, each of which corresponds to a local modification of the candidate solution. A simple local search process, referred to as *iterative improvement*, starts at a randomly generated "seed" and, at each decisions point, chooses the neighbor of the current node that most improves the current evaluation score.

Many variants of local search techniques were tried within GENH. We present here a simplified version that was used in the reported experiment; this specific algorithm is shown in Figure 1. First, GENH generates an initial seed by randomly selecting values for each weight (w_i) and then normalizing them to sum to 1. This heuristic is then evaluated. GENH's evaluation of a candidate heuristic consists of invoking the APA scheduler on the problem instance to perform greedy search using the candidate heuristic, and then scoring the resulting schedule

with the given objective function (accomplished by the `eval` function in the figure).

In GENH's search space, the neighbors of a node are generated by adjusting the weight of a selected single attribute and then dividing each weight by the sum of the weights so that the resulting vector sums to 1 (accomplished by the `neighbors` function in the figure). The best neighbor resulting from tuning one attribute is the starting point for the adjustments made to the next selected attribute, until all the attributes have been singly tuned. In the reported experiment, the attributes are processed in a random order.

The set of adjustments to the selected weight is based on two parameters: a *boundary* parameter which determines the maximum adjustment (increment or decrement) to the weight's current value, and a *granularity* parameter which determines the set of adjustments (uniformly distributed) between these bounds. After each attribute has been tuned once, the granularity and boundary parameters are decreased, and the entire set of attributes is tuned again. On each iteration a smaller neighborhood of the current best heuristic is searched with a finer granularity. The search terminates when the granularity parameter exceeds user-specified minimum.

In the reported experiment, the granularity parameter is initialized to 0.1 and the boundary parameter is initialized to 0.5. If the initial value for the weight selected for tuning is v, the set of neighbor values considered is $\{v_n = \max(0, v - 0.5) + k \times 0.1 \mid v_n \leq \min(1, v + 0.5)\}$, where k is an integer. After each iteration, both parameters are divided by 5; hence, there is a constant upper bound on the number of neighbors. Thus in the second iteration, the set of neighbor values is $\{v_n = \max(0, v-0.1)+k \times 0.02 \mid v_n \leq \min(1, v+0.1)\}$. In the reported experiment, the search was terminated when the granularity parameter is less than 0.004 (*i.e.*, after three iterations of the repeat loop).

The HBSS Technique

In this section, we describe the *Heuristic-Biased Stochastic Sampling* (HBSS) algorithm, first introduced in [2]. Within HBSS, the desired balance between heuristic adherence and exploration in the search space is determined by specifying a *bias function* and a *ranking function*. Both of these functions enable the user to encode information about the heuristic function; this additional knowledge is employed during search to tailor the guidance provided by the heuristic.

The ranking function enables the encoding of information regarding what relevant distinctions to make within the range of the heuristic function. The ranking function partitions the heuristic's range into equivalence classes and determines the magnitude of the quality differences between classes. The bias function enables the encoding of information regarding the accuracy of the heuristic and the amount of exploration that is desired. A stronger bias tends to follow the heuristic's advice more often and a weaker bias tends to explore farther off the greedy trajectory in the search tree.

```
HBSS-iterate(root, heuristic_fnc, rank_fnc, bias_fnc)
begin
current_state = root
nodes = successors(root)
repeat until empty(nodes)
    current_state = HBSS-select(nodes,
            heuristic_fnc, rank_fnc, bias_fnc)
    nodes = successors(current_state)
endrepeat
return current_state
end

HBSS-select(nodes, heuristic_fnc, rank_fnc, bias_fnc)
begin
score nodes with heuristic_fnc
rank nodes with rank_fnc based on scores
weight nodes with bias_fnc based on ranks
normalize weights to yield selection probabilities
select node stochastically according to probabilities
return selected node
end
```

Figure 2: `HBSS-iterate` performs one iteration of the HBSS algorithm, and `HBSS-select` is used at each decision point within `HBSS-iterate`; where `successors` returns the states resulting from application of each observation enabled in the given state.

The algorithm for generating one sample, or schedule, is given in Figure 2; this version assumes that the search tree has finite branching and finite depth, as is the case for our domain. At each decision point, the alternative choices are scored according to the given heuristic function, and each choice is assigned a rank, according to the given ranking function, based on these scores. We assume that ranks are positive integers and that the top rank is 1. The given bias function is then used to assign a non-negative real-valued weight to each choice based on its rank. The assigned weights are then normalized by dividing each one by the sum of the weights. The normalized weight for an alternative choice represents its probability of being selected; a choice is selected according to these probabilities by a weighted stochastic process.

A typical ranking function is one that puts choices with equal heuristic scores into the same equivalence class and then assigns these classes consecutive integers. A typical family of bias functions are the *polynomial bias functions* defined as $\texttt{poly}_n(r) = r^{-n}$, where r is the rank and n is the polynomial degree.

Figure 3 illustrates how `HBSS-select` works. In this hypothetical example, at each decision point there are three choices which are assigned unique ranks. The table in the figure shows how the three heuristic scores for an example decision point are converted into three selection probabilities by the steps in `HBSS-select` using a $\texttt{poly}_1$ bias function; in this example, lower scores are better. Each node in the tree is labeled with its assigned

Heuristic:	100.25	150.67	300.04
Rank:	1	2	3
Bias ($1/r$):	1.00	0.50	0.33
Probability:	0.55	0.27	0.18

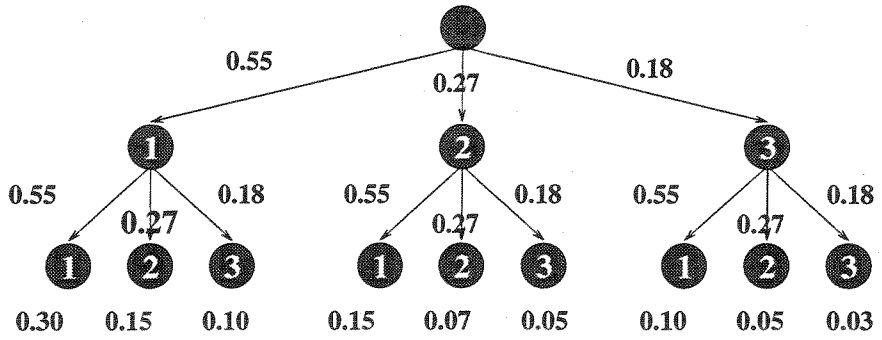


Figure 3: Example of solution probability distribution induced by `HBSS-iterate` with a $\texttt{poly}_1$ bias of $1/r$ on an hypothetical tree. In the tree, nodes are labeled with the assigned rank, arcs are labeled with the selection probability, and the bottom numbers indicate the probability of reaching the corresponding leaf node.

rank and each arc is labeled with the resulting selection probability. The number below each leaf node of the tree is the probability of reaching that leaf, which is the product of the selection probabilities of all the arcs in the path from the root node to the leaf node. Hence, the incremental application of `HBSS-select` induces a probability distribution over the space of possible solutions.

Typically, the `HBSS-iterate` routine is executed some number of times, based on the available schedule generation time, and the best schedule with respect to the given objective function found is returned. It is difficult to predict which bias function is going to yield the best performance on a given problem, with a given heuristic and a given number of samples. One approach to ameliorating this problem is to use more than one bias function and alternate on each sample. This would allot an equal number of samples (within 1) to each bias function regardless of the number of total samples. Within the APA scheduler, a "sample proportion" can be specified for each bias function so that uneven computation shares can be allotted. In the reported experiment, we use this multi-bias HBSS approach with $\texttt{poly}_2$, $\texttt{poly}_3$, and $\texttt{poly}_4$, each getting an equal share of samples.

Experiment Methodology

In this section, we describe the methodology of the computational experiments, which were carried out within the observation scheduling problem class. One of the experiment's goals is to test the hypothesis that adaptive, stochastic-based scheduling significantly improves the quality of the schedules over a non-adaptive, deterministic approach in the telescope observation domain. A second goal is to empirically measure the share of the improvement yielded by GENH and HBSS individually, as well as to ascertain whether their combined use is better than either single technique.

To test our hypothesis, one of the scheduling methods employed is a non-adaptive, deterministic technique based on ATIS dispatch (defined in the second section). The dispatch policy was simulated as greedy search with the following "dispatch heuristic": $0.625 \times \texttt{priority} + 0.3125 \times \texttt{run count} + 0.0625 \times \texttt{enablement time}$. The attribute *run count* is the number of times the observation occurs in the current schedule prefix, and the attribute *enablement time* is the time currently remaining in the observation's enablement interval. Lower priority numbers indicate more importance, and lower values are also preferred for the run count and enablement time attributes; hence, a lower heuristic score is better.

For the test cases, we selected a set of problem instances over the 251 day interval of Julian Dates $[2450400, 2450650]$. This range of instances assures variation in problem characteristics (specifically, with respect to load capacity and phase of the moon). The objective function is the weighted summation of two attributes: one which penalizes enabled observations that are missed in the schedule based on their priority, and one which prefers schedules that minimize the average airmass of the scheduled observations. The priority penalty is computed as $\sum 10^{Pmax-P(mo)}$, where the summation is over all missed observations mo, $P(mo)$ is the priority of observation mo, and $Pmax$ is the maximum priority in the observation set. The average airmass is calculated as $\sum A(so) \times D(so) \;/\; (dawn - dusk)$, where the summations are over all scheduled observations so, $A(so)$ is the airmass of so, and $D(so)$ is the duration of so. Airmass is computed based on the secant of the telescope pointing angle.

The weights assigned in the objective function are 100 for the priority penalty and 1 for the average airmass. Given the ranges of these two attributes, the attribute weights ensure that if one schedule has a better score

with respect to the priority penalty attribute than another, then it will have a better objective function score regardless of the average airmass score; hence, the airmass attribute will only break ties between schedules that have the same priority penalty score.

The set of candidate attributes for GENH includes the attributes contained in the dispatch heuristic, as well as an additional attribute, *airmass*. The reason for including the additional attribute is to exploit GENH's ability to examine arbitrary collections of attributes and to weed out noneffective ones.

We carried out a comparative analysis of the following four scheduling techniques:

- *Greedy Dispatch*: Deterministic greedy search using the dispatch heuristic
- *Greedy GenH*: Deterministic greedy search using the problem's GENH heuristic
- *HBSS Dispatch*: Multi-bias HBSS using the dispatch heuristic
- *HBSS GenH*: Multi-bias HBSS using the problem's GENH heuristic

The comparisons of interest are the following: (*i*) Greedy GENH *vs.* Greedy Dispatch, (*ii*) HBSS Dispatch *vs.* Greedy Dispatch, (*iii*) HBSS GENH *vs.* Greedy GENH, and (*iv*) HBSS GENH *vs.* HBSS Dispatch.

On each problem instance in the test suite, the performance of each of the four techniques was evaluated as follows. For the two deterministic techniques, the performance evaluation is the objective function score of the single schedule generated. For the two stochastic sampling methods, five runs of 15 samples each were performed. Hence, the $\texttt{poly}_2$, $\texttt{poly}_3$, and $\texttt{poly}_4$ bias functions are each employed on 5 samples. For each run, the best objective function score from the 15 samples is collected; the performance evaluation is the average of these five best scores.

Interpretation of Results

In this section, we discuss the empirical results gathered thus far in our ongoing investigation. The performance results for the four above comparisons are illustrated in Figures 4 and 5. Rather than illustrating the comparisons in terms of the objective function scores, they are illustrated with respect to each of the unweighted attribute scores, in order to more closely examine the differences between the four techniques. Since the weighted priority penalty always dominates the weighted average airmass, the performance w.r.t. the objective function score exactly mirrors the performance w.r.t. the unweighted priority penalty. Figure 4 illustrates improvement w.r.t. priority penalty, and Figure 5 illustrates the improvement w.r.t. average airmass. Since lower scores are preferred, improvement of technique *A over* technique *B* is computed as *B*'s score − *A*'s score, and a negative improvement indicates the improvement of *B over A*. Negative improvement is utilized in Figure 5; however, since the `y-axes` in Figure 4 are logarithmic, only positive improvement is plotted.

The two plots in the first (top) row of Figure 4 show that for all but 3 of the 251 problem instances, the heuristic generated by GENH significantly outperformed the dispatch heuristic when both were used in greedy search. For the three exceptions (JDs 2450480, 2450481, and 2450543), either an equally good (or better) heuristic did not exist in GENH's search space $\mathcal{S}$, or else GENH failed to find one within the limitations of the specific search strategy employed.

The two plots in the figure's second row show that, on every problem instance, HBSS with the dispatch heuristic was able, in only 15 samples, to find a significantly better schedule than the greedy solution. Comparing the first two rows indicates that both HBSS and GENH individually almost always achieve the same degree of improvement over greedy dispatch.

The bottom two rows in the figure illustrate how the combination of HBSS and GENH compares to each technique individually. The results plotted in the third row illustrate how much HBSS helps GENH, and the results plotted in the fourth row illustrate how much GENH helps HBSS. The right plot in the third row shows that on 5 problems, HBSS was not able to find, within 15 samples, a schedule as good as the one GENH found. The left plot in that row shows that HBSS did not often help GENH – it did so on 13 of the 251 problems; however, it significantly helped on the three problems for which GENH did much worse than greedy dispatch. This is an indication that HBSS and GENH are complementary.

The results plotted in the last row indicate that GENH helped HBSS on 87 problems, hurt HBSS on 12 problems, and had no affect on the remaining 152 problems. This suggests that giving HBSS a head start with a better heuristic can help it find better schedules, but that HBSS' performance is not strongly dependent on the quality of the heuristic – which was one of the intended features of HBSS' design! We expect that the greater the number of samples used by HBSS, the less sensitive it will be to heuristic inaccuracy.

We now turn our attention to the secondary scheduling objective: minimizing airmass. The four comparative analyses with respect to the average airmass attribute are shown in Figure 5. The two plots in the figure's top row illustrate the performance of HBSS and GENH individually, and the two plot in the bottom row illustrate the performance of the two techniques in combination.

The right plot in the top row indicates that HBSS gains the improvement over greedy dispatch with respect to the priority penalty at the cost of increased airmass. In contrast, GENH makes this tradeoff much less often and, furthermore, GENH usually gains improvement over greedy dispatch as well.

The left plot of the figure's bottom row shows that HBSS, with 15 samples, yields only minor airmass improvement over greedy search when both use the GENH heuristics. However, the right plot in that row indicates that GENH's airmass improvement transfers to the HBSS context, greatly improving HBSS with respect to this secondary attribute. These results, conjoined with the re-

sults (shown in the bottom row of Figure 4) that GENH can sometimes help improve HBSS' priority scores, indicate that HBSS and GENH are complementary.

Figure 6 illustrates GENH's output, *i.e.*, the heuristic attribute weights, over the test problem class. The sparseness of the plots for the heuristic attributes for priority and enablement time indicate that often GENH chose a zero weight, thus eliminating the attribute from the heuristic. Elimination from the heuristic happened on 173 of the 251 problems for the priority attribute and on 162 problems for enablement time; whereas it only happened twice for the airmass attribute and only once for run count. The average weights for the four attributes are as follows: airmass: 0.595, run count: 0.296, priority: 0.061, and enablement time: 0.048.

These results help explain why GENH does so well with respect to the airmass objective; however, they are very surprising with regards to the primary objective of priority. How is GENH able to improve over dispatch with respect to the priority objective when dispatch assigns the priority attribute the highest weight and GENH places such little importance on it? Recall that the priority objective attribute differs from the priority heuristic attribute. The latter can be easily evaluated on either a complete or partial schedule since it is defined in terms of what is included in the schedule. Whereas the former is the more global property of what is missing from the (complete) schedule. It is possible, due to interactions among the attributes and specifics of the observation request set, that a scheduler which locally optimizes with respect to airmass generates schedules that also score well with respect to the global priority objective.

Conclusion

In this paper, we described two approaches to improving schedule generation with respect to the domain-specific objectives. GENH improves schedule quality by improving the search heuristic, and HBSS improves schedule quality by making better use of a given search heuristic. The results demonstrate the superiority of an adaptive scheduling approach and suggest that HBSS and GENH are complementary in the following respects:

- HBSS can compensate when GENH misses the mark and produces an heuristic worse than dispatch.
- GENH enables HBSS to improve the priority score over dispatch without sacrificing airmass and, thus, to optimize both the primary and secondary scheduling objectives.

When employing HBSS in practise, the intent is to first perform a greedy search and then to run HBSS for the remaining available solution time in an attempt to find a better solution *via* informed exploration. In addition, within the APA scheduler we can use any combination of the four solution strategies discussed above; our current meta-strategy is as follows: first, greedy dispatch, second, greedy GENH, and third, n samples of HBSS GENH for the remaining allotted solution time.

We intend to explore other search strategies within GENH, *e.g.*, a multistart approach in which multiple GENH iterations are run, each with a different random seed. We also intend to investigate how the number of HBSS samples affects the relationship between GENH and HBSS. Furthermore, we have begun to address multi-night objectives, *e.g.*, resource allocation fairness, and the leverage over dispatch achieved by the combination of HBSS and GENH should become even more significant.

Acknowledgments

Thanks to Stuart Rodgers, who played a major role in the original development of GENH. Thanks also to the APA project team's former members: Mark Drummond, Keith Swanson, and Ellen Drascher, and our collaborators: Gregory Henry (Tennessee State University), Donald Epand (Fairborn Observatory), and Louis Boyd (Fairborn Observatory).

References

[1] Boyd, L., Epand D., Bresina J., Drummond M., Swanson K., Crawford D., Genet D., Genet R., Henry G., McCook G., Neely W., Schmidtke P., Smith D., and Trublood M. 1993. Automatic Telescope Instruction Set 1993. In *I.A.P.P.P. Comm.*, No. 52. Oswalt (ed).

[2] Bresina, John L. Heuristic-Biased Stochastic Scheduling. In *Proceedings of AAAI-96*, Portland, OR.

[3] Bresina J., Drummond M., Swanson K., and Edgington W. 1994. Automated Management and Scheduling of Remote Automatic Telescopes. In *Optical Astronomy from the Earth and Moon*. ASP Conference Series, Vol. 55. D.M. Pyper and R.J. Angione (eds.).

[4] Bresina, J., Edgington, W., Swanson, K., and Drummond, M. 1996. Operational Closed-Loop Observation Scheduling and Execution. In *Working Notes of the AAAI Fall Symposium, Plan Execution: Problems and Issues*. Cambridge, MA. Available in *AAAI Technical Report FS-96-01*.

[5] Drummond, M., Bresina, J., Edgington, W., Swanson, K., Henry, G., and Drascher, E. 1995. Flexible Scheduling of Automatic Telescopes over the Internet. In *Robotic Telescopes: Current Capabilities, Present Developments, and Future Prospects for Automated Astronomy*. ASP Conference Series, Vol. 79. G.W. Henry and J.A. Eaton (eds).

[6] Henry, G.W., 1996. ATIS dispatch scheduling of robotic telescopes. In *New Observing Modes for the Next Century*. ASP Conference Series, Vol. 87. T. Boroson, J. Davies, and I. Robson (eds).

[7] Morris, R.A., Bresina, J.L., and Rodgers, S.M. 1997. Automatic Generation of Heuristics for Scheduling. In *Proceedings of IJCAI-97*. Nagoya, Aichi, Japan.

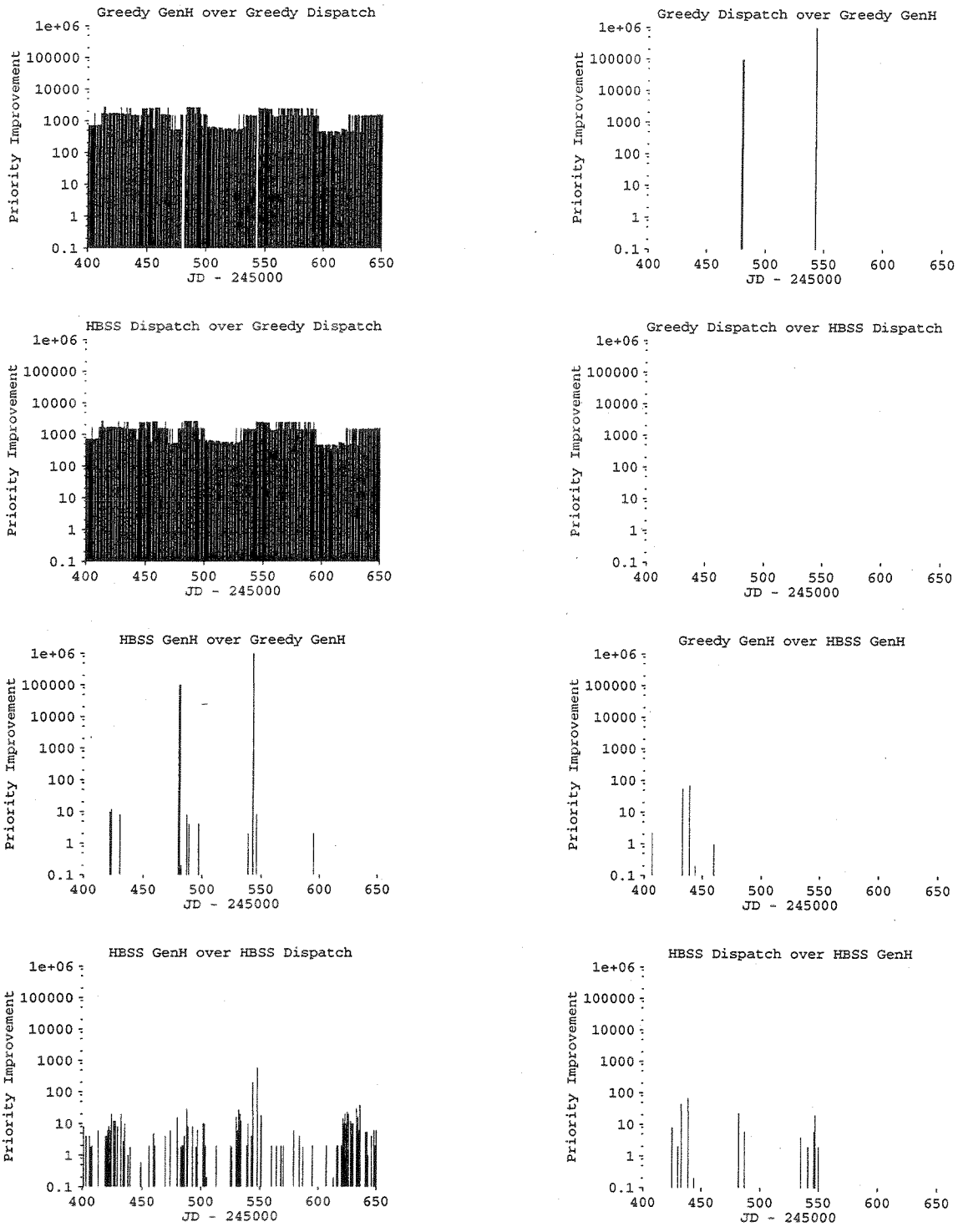


Figure 4: Improvement with respect to the priority attribute score; HBSS performance is based on the best score after 15 samples averaged over 5 runs. Note, the **y-axis** is log scale and, hence, only positive improvement is plotted.

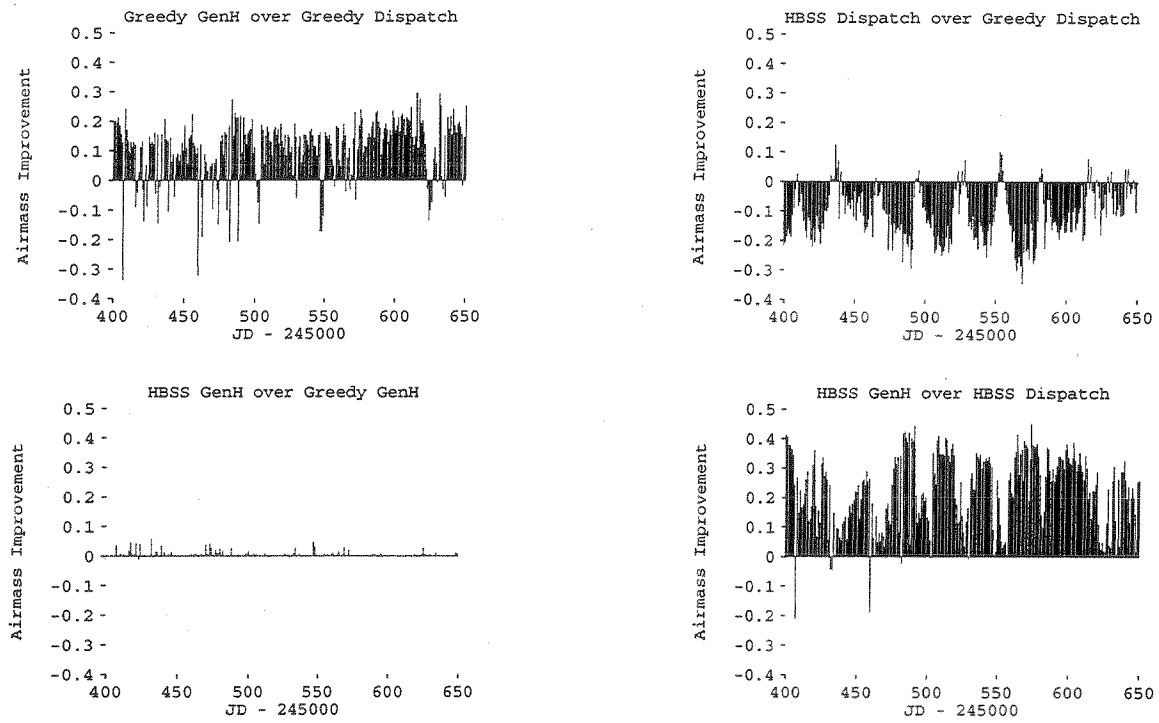


Figure 5: Improvement with respect to the airmass attribute score; HBSS performance is based on the best score after 15 samples averaged over 5 runs.

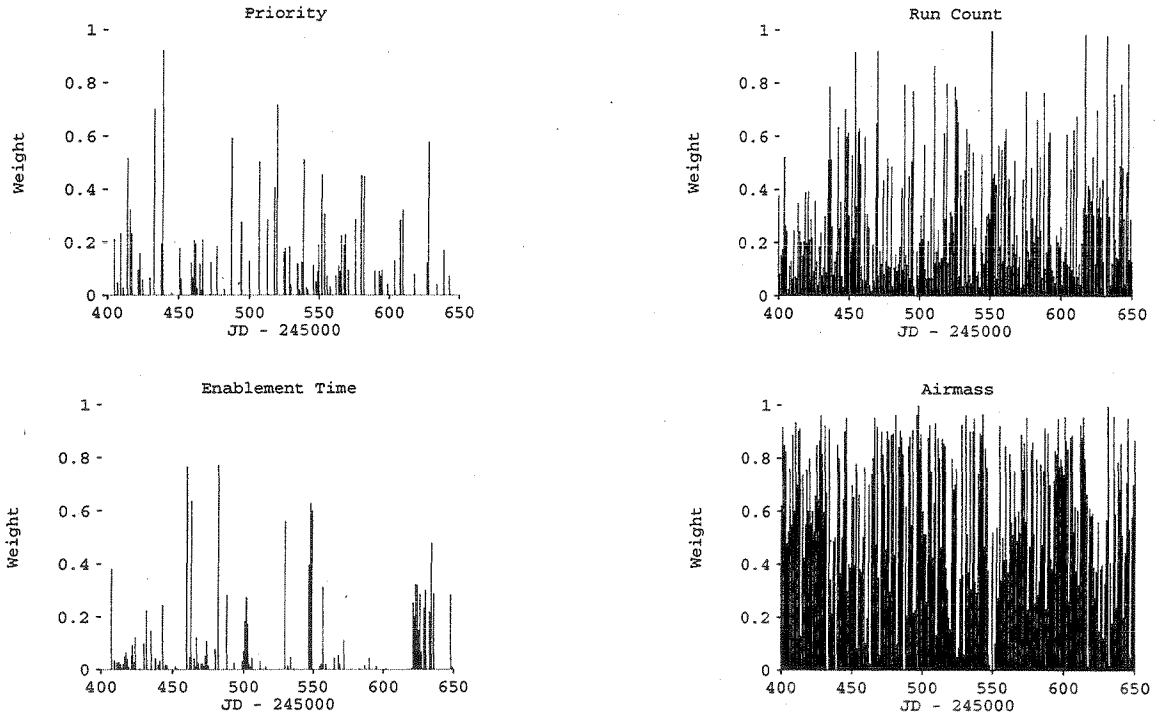


Figure 6: Weights generated by GENH for each of the four candidate attributes.

An Object-Oriented Scheduling Architecture for Managing the Data Relay Satellite Requests

Amedeo Cesta
IP-CNR
National Research Council
Viale Marx 15
I-00137 Rome, Italy
amedeo@pscs2.irmkant.rm.cnr.it

Paolo Bazzica and Gianni Casonato
Corso di Laurea
Ingegneria Informatica
Università di Roma "La Sapienza"
I-00198 Rome, Italy
{bazzica, casonato}@dis.uniroma1.it

Abstract

In this paper an automated system is described for managing the daily activities of the DRS satellite system. In the system the object-oriented and artificial intelligence methodologies have been jointly used to develop a comprehensive approach to the problem. Particular attention has been given to the representation of the scheduling domain, the dynamic maintenance of a solution, and to the interaction with different types of users. The system offers flexible services and its performance is acceptable for the operative environment.

Problem Description

The Data Relay Satellite (DRS) System is an European Space Agency program aimed at providing a data relay service between Low Earth Orbiting (LEO) satellites and their ground terminals (ESA 1989; 1990). Actually this program is in the last step of development, and it will be operative within 1999.

The DRS infrastructure (see Figure 1) consists of a constellation of two satellites and in a set of ground stations that allow:

- almost total coverage area;
- strong reduction of the LEO's ground terminal network;
- reduction in data-distribution problems;
- reduction of required on-board data storage capacity for LEO satellites.

The scheduling problem of DRS consists in the production of a mission plan, that allow the clients to utilize the transmission services. An high number of access requests is expected, so that their temporal extension exceeds the total transmission time available, introducing conflicts that have to be solved following some quality objectives.

Given the technical characteristics of the DRS system, the crucial aspect in the production of the plan is the management of the link between the DRS and the LEO satellites, while the links between DRS and ground stations are less problematic. The first type

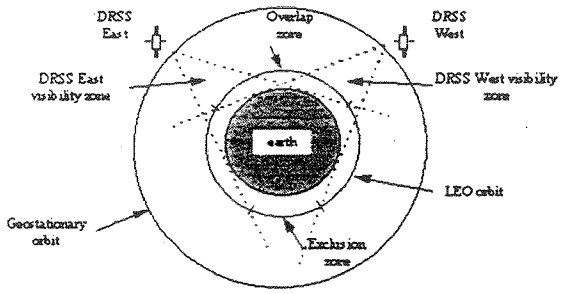


Figure 1: The Data Relay Satellite System

of link imposes the satisfaction of physical constraints of the DRS's antennas, temporal constraints of the requests, and requirements of priority, commercial value and allocation preference.

To summarize the problem addressed, we use the usual scheduling terminology (French 1982) identifying problem's resources, activities, activities constraints, and optimum criteria. It is worth noting that from now on we speak of requests instead of activities, due to the particular application domain.

Resources. The physical system modeled by the application consists of one out of the two DRSs, composed of two Inter Orbit Links (IOL). Consequently, a request can be allocated if an interval of time can be established in which an antenna of the IOL is available.

Requests and related constraints. All user requests specify a number of desired characteristics which include:

- static priority associated to the request's owner;
- technical requirements: these include the band, speed of transmission and the number of channels required;
- user flexibilities: minimum and maximum time intervals for the duration, the interval of time within which the access must be scheduled (flexibility interval) and the utility function associated with these flexibilities (see Figure 2);

- user preferences: preferred values for the duration and the actual time of access (see functions f_1 and f_2 in Figure 2).

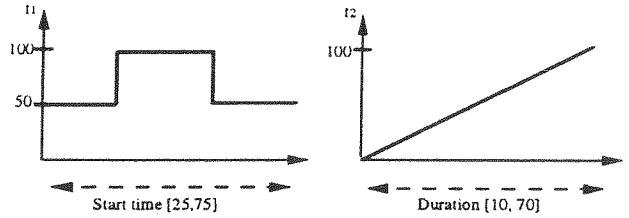


Figure 2: Temporal flexibility and preference functions

Optimization. The goals which need to be satisfied in generating the Detailed Assignment Plan (DAP) are:

- scheduling of as many access requests as possible;
- satisfaction of as many user preferences as possible;
- priority for preferences of requests having a higher 'relevance' coefficient.

The goals are potentially conflicting: an optimization in resource use required to satisfy the first goal would imply taking full advantage of user specified flexibilities but in doing so, the preference (or utility) function given by the users may not be satisfied. The other two goals are in turn partially contrasting, since maximizing user preferences does not necessarily coincide with satisfying the requests of preferred users.

According to the technical documentation, the production of the DAP is supposed to follow an iterative process repeated three times, and that involves two types of human operators:

- *Commercial operators* at the Mission Control Center: negotiates with the clients the sale of the free transmission spaces, and inserts in the plan the related activities;
- *Spacecraft engineers* at the Operation Control Center: modifies the plan inserting in some special activities for the maintenance of the system operativity, and requests with a special requirement of urgency.

These two operational profiles follow different and potentially conflicting objectives (maximum satisfaction of requests vs. DRS's resources saving). Those objectives have to be integrated together in an automated scheduling system that supports decision making in this environment.

Scheduling Architecture Design

Artificial Intelligence (AI) techniques provide an approach to a planning and scheduling problem which is based on three fundamental aspects (Figure 3):

- representation of the domain and solution management;
- generation of satisfactory or optimal solutions;
- interaction with the user.

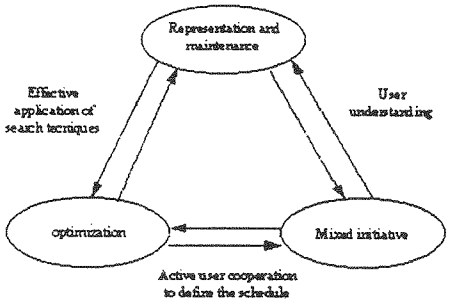


Figure 3: Basic aspects of the AI approach

These three aspects may increase the project complexity for a system that should supply all these characteristics. This problem can be solved using the tools provided from the object-oriented analysis and design techniques.

The representation of the domain has to be dynamic, able to follow the physical changes in the domain and to supply an incremental building of the plan. In the same time has to be symbolic to allow the user high level understanding. The need to product optimal plans, for the high complexity of the scheduling problems, claims the use of heuristics techniques of search. These techniques quickly generate a solution, and then, if necessary, allow the user to directly modify the building process. The ideas of decomposition, abstraction and hierarchy help to individuate the atomic entities of the problem. Each of those entities is in relation (inheritance, aggregation and use) with some other entities at the right abstraction level. We have used object-oriented methodology to realize an architecture for representation and maintenance of a scheduling domain. This architecture, named O-OSCAR (Object-Oriented Scheduling Architecture), can supply all the tools needed for the optimization task and for the interaction needs of the system. At a very abstract level O-OSCAR represents the four basic entities in a scheduling problem (similarly to (Wolf 1994)):

- processors;
- operations;
- decisions;
- constraints.

The complete O-OSCAR architecture is organized in a hierarchical structure, subdivided in two levels: abstract level and concrete level.

The abstract level defines entities with general characteristics and functionalities, common in all scheduling problems. The concrete level contains the concrete

entities, derived from the abstract ones, that describe a more specific set of scheduling problems. Following the objective of the maximum applicability of the architecture, the class of satellite scheduling problems (SSP) has been defined. This class contains the DRS problem, and it is constituted by a general job-shop with additional quantitative precedence constraints and allocation preference for the operations. The concrete entities can be instantiated to a real problem (to DRS problem in our case) and by using a transaction model of utilization —based on insert, delete and retrieve actions— they can support the maintenance of a solution as an incremental building strategy.

Figure 4 shows, in Booch notation (Booch 1994), the basic entities in the object-oriented design of both the abstract level and the concrete level.

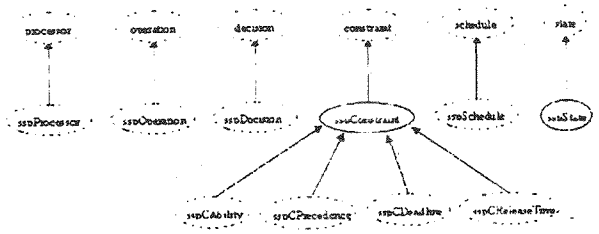


Figure 4: Basic entities in O-OSCAR design

A Scheduler for the DRS

Having implemented a complete architecture able to represent the knowledge about a scheduling domain, we use it to realize a complete activity sequencer for the described satellite system.

The main objective of the project is to keep the user inside the scheduling process. For this reason a study of the scheduling environment had to be performed to define the aspects of the work that mainly need an automatic support. The DRS planning environment supports the two typical users introduced above: the commercial user at the the Mission Control Center and the operative user (spacecraft engineers) at the Operation Control Center.

For each type of users a personalized set of instruments to manipulate the schedule had been defined. This required the definition of two User Profiles. In the DRS sequencer all the aspects common to the two profiles form the kernel of the sequencer, while those typical of a specific profile are isolated in a series of separated modules. The object-oriented design made simpler the adaptation of the system to the needs of different users.

The process of schedule building is at each moment under the user's control. In each planning phase the user has the ability to operate directly on the schedule, if needed. A direct consequence of this user needs is the project choice to symbolically represent the knowledge of the domain. The nature of the defined objects allows the user manipulation at an abstraction level near to the highly symbolic human reasoning. The use of the C++ language, that directly supports the object-oriented decomposition, allows to achieve an high execution efficiency.

The sequencer architecture allows the integration of new heuristics while minimizing the effects on the code. At present the sequencer provides two scheduling heuristics: the first is strictly tied to the problem and allows the construction of the schedule managing the backtracking in a dependency directed way (a greedy heuristics), the second uses a generalized simulated annealing technique to perform the same task. The user has the ability to interactively change the heuristic to be used in each phase of the scheduling process. The availability of a heuristics set claims for the definition of an evaluation method capable to compare the results obtained with different heuristics (or even those due to a same heuristic used with different tuning). In the case of the DRS, the quality evaluation of a schedule is connected to the existence of a set of user-selectable criteria able to evaluate the schedule quality, according to a set of scheduling objectives. The flexible way to do that is by defining a set of functions that associate a quality value to each activity in any particular schedule. The quality of the whole schedule is then defined as the sum of the activities qualities that form it.

In the DRS scheduler three quality functions exist:

- A global quality function that, for each scheduled request, defines a real number directly dependent on (1) the priority of the request, (2) the commercial value of the request (3) the response to the user wishes for the start and the duration of the request.
- A priority function that, for each scheduled request, defines a real value directly dependent on the priority of the request. This function rewards the schedules that accepts the requests with the highest priority.
- A flexibility function that rewards the schedules which better respect the user defined preferences about the start time and the duration of each request accepted.

The choice of define a set of functions instead of just one is due to the need to evaluate the schedule quality from different viewpoints. In general terms, the user has the ability to choose one of these functions to define the scheduling objective. Each time the control of the scheduling process is transferred to the system the schedule will be manipulated so that the selected quality function is maximized. To direct the user towards the best choice with respect to the particular problem involved a comparison criterion between the available heuristics is needed. To this purpose the DRS sequencer evaluates the schedule quality using the Expected Solution Quality (ESQ) theory (Bresina *et al.*

1995). This method has been formulated to statistically evaluate the performances of the scheduling algorithms. By building a set of random-generated solutions a gaussian density quality function is inferred and the standard deviation of the scheduling algorithm used from the mean of this function is used to define a measure of the algorithm quality. The DRS sequencer incorporates an interactive version of the ESQ algorithm able to perform the quality of a partial schedule in each moment of the planning process.

The block diagram of the software system is presented in Figure 5. The modularity of the system, due to the project choice of facing the aspects of knowledge representation, solution optimization and mixed initiative in independent software modules, allows the definition of new heuristics and/or new user profiles without modifying the main structure of the system.

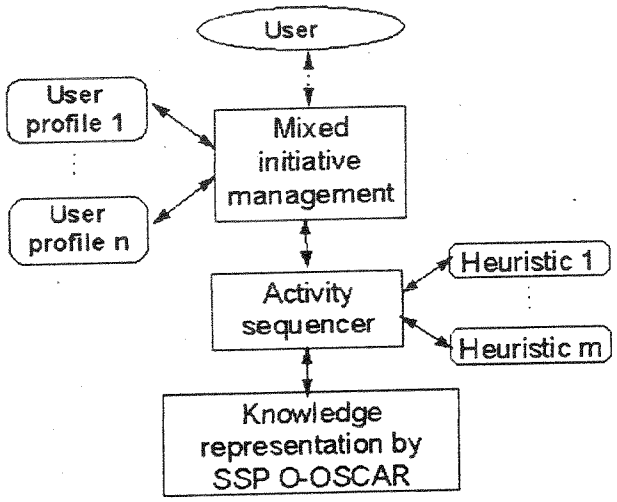


Figure 5: DRS O-OSCAR architecture

The Module named SSP O-OSCAR in Figure 5 is responsible for the representation of the domain and for the maintenance of the current solution. To represent the temporal aspects of the solution the set of dynamic propagation algorithms described in (Cervoni *et al.* 1994; Cesta and Oddi 1996b) are used.

The system can configure its own level of initiative following the user wishes. Its role in the scheduling task can simply consist of supporting the user in his effort by maintaining the information about the schedule and signaling when the user choices conflicts with previous ones. Otherwise the system can try the insertion of a set of requests, specified by the user with the desired heuristic. Finally the system can generate a complete schedule from a set of specified requests allowing a user post-initiative on the schedule.

The choice to implement an interactive system requires the use of a graphical instruments set to interact effectively with the user. This choice imply the risk to compromise the portability of the system due to the incompatibility of the main hardware platforms regarding the graphic interface management. The DRS sequencer give solution to this problem entrusting on the AMULET library, developed in C++ at the Carnegie Mellon University (Myers). This library, allows the creation of a graphical interface abstracted from the hardware by defining a set of instruments like windows, buttons etc. The concepts of object-oriented programming like inheritance, polymorphism and encapsulation are fully implemented allowing the coding of the interface at a very high level of abstraction. The fact that the sequencer is written in C++ causes the availability of the system, by simply recompiling on all the platforms supported by AMULET. At present these are a big set of UNIX/Xwindow flavors, Windows 95/NT and the MacOS.

In building the sequencer the mixed initiative model based on mutual constraining of behavior (Tate 1997) has been followed. The key point of this approach is the choice to share a common plan model between the users; on this model each user operate the desired manipulations. The system interprets each user action as an attempt to constraint the final aspect of the plan and check the feasibility of the proposed modification.

In the DRS system the user can be the commercial operator, interacting with the commercial-profile sequence, or the spacecraft engineer, interacting with the operative-profile sequencer. Each one of those users contributes to the final aspect of the plan by proposing the allocation of activities and by reacting to the changes due to the actions of the other users. All the users share a common, object-oriented vision of the scheduling domain that defines the scheduling vision and interact with the system via a personalized set of instruments. This view of the scheduling process is reported in Figure 6. The two scheduling clients, the commercial one and the operative one, are discussed in the following subsections.

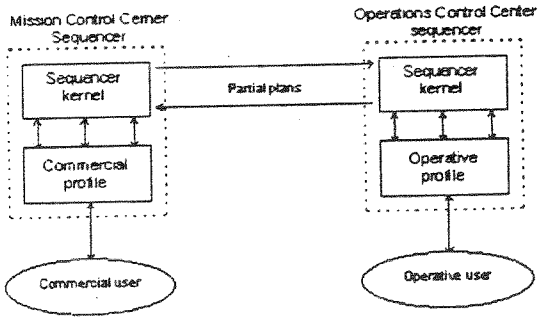


Figure 6: DRS O-OSCAR users views

The Commercial Profile

The commercial version of the sequencer is directed to the user of the Mission Control Center. The main screen consists of two windows, the main window and the Gantt chart window.

In the main window, Figure fig:comm, there are three lists with the information about the service requests proposed to the system, the requests accepted by the system and the requests refused by the system. The system allows loading and saving of partial schedules on the disk, the possibility to input a new request interactively via a dedicated mask and the capability to load a set of request from a disk file. The scheduling process consists on the selection of a request set and in the successive attempt to schedule it through the selection of the button Process. The interesting key points of this profile are:

- the attempt to abstract from the technical detail of the task;
- a simplified view of the scheduling task;
- exclusive tools to trade with the clients about the parameters of the requests refused by the system.

Regarding the last point this version of the sequencer provides the capability to select a set of requests from the refused requests list and the management for each request of the relative conflicting parameters. At this purpose is useful to use the Gantt chart to be able to directly the availability of the temporal parameters.

The Gantt window , Figure fig:gantt, allows the selection of each box associated with a request to have access to the relative detailed parameters. The successful trading of a request causes the apparition of the relative box in the Gantt chart.

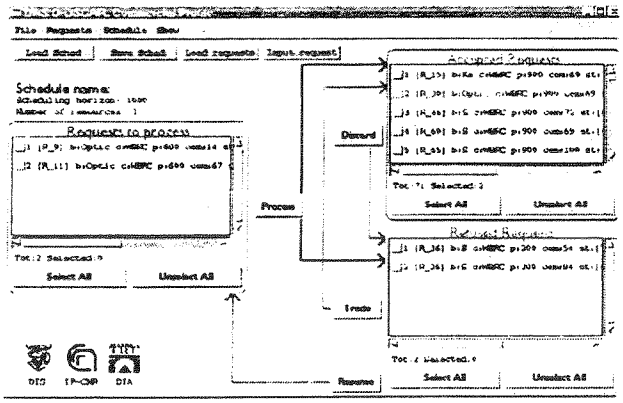


Figure 7: The interface for the commercial user

The Operative Profile

The operative version of the sequencer has the objective to incorporate in the partial plans created by the mission control center the requirements that allow

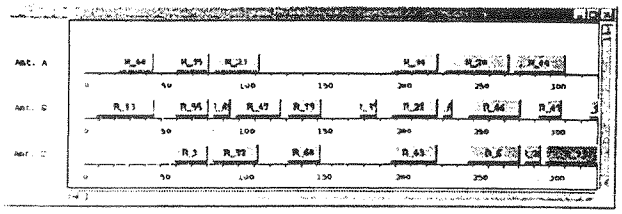


Figure 8: Current solution representation through a Gantt chart

the operativity of the orbiting platform (e.g. orbital shifts, maintenance of antennas), manage the available resources (e.g. excluding a broken antenna from the schedule) and allow the input of "last minute" requests (e.g. emergency operations). The need to minimize the effects of those operations claims full access to the available scheduling strategies provided from the system. When the program is started the screen shows the next window and the Gantt. The Gantt chart window is the same provided from the commercial profile sequencer.

In the main window of the Operative Profile, shown in Figure fig:oper the following information is continuously represented:

- The global state of the schedule.
- The total number of requests proposed to the system.
- The total number of rejected requests.

The details relatives to each of these aspects are accessible by the "More Info..." buttons. A dedicated button allows the use of the ESQ to evaluate the actual schedule quality (as explained below). The key points of this profile are:

- the possibility to alter the scheduling domain by removing an antenna;
- the ability to change the heuristic used to schedule the selected requests.
- the direct management of the backtracking.

As far as the last point is concerned the user has the capability to choose directly the activities to remove in case they are in conflict with the selected activity. In this way the modifications due to the backtracking are under the direct control of the user, allowing the minimum side effect of the reactive phase on the existent partial schedule.

System Performances

Two aspects of the system performance have been considered:

- the quality of the generated schedules;
- the time needed to generate the schedule.

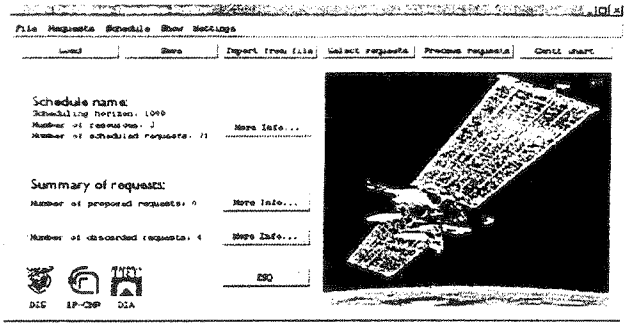


Figure 9: The interface for the operative user

As far as the first aspect is concerned, a version of the ESQ algorithm (Bresina *et al.* 1995) has been implemented which allows to compare the system solution with the average quality of randomly generated solutions.

This comparison is possible according to different quality criteria as shown in the Figure 10 in which the performance w.r.t. three aspects, flexibility, priority and global quality, are shown.

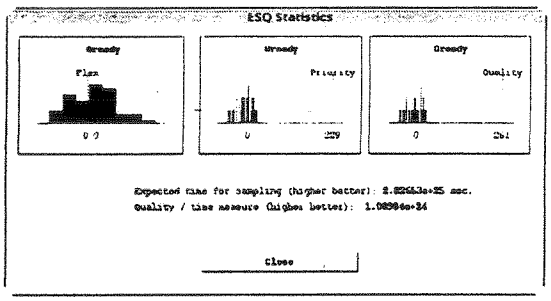


Figure 10: The ESQ evaluation of a solution

The system had be tested with the DRS problem simulator developed during a previous approach to the problem (Adinolfi and Cesta 1995). A typical daily DRS problem consists in 75 requests and it is integrally processed by the system in times compatible with the interactive use of the software. The 75 requests problem requires approximately 1 minute on the Windows 95 version of the sequencer with a Pentium at 75 MHz. The times on a typical UNIX workstation are reduced by one half. Figure 11 shows the CPU time (in seconds —y axis) needed to produce a solution for problems of increasing dimensions (number of requests —x axis) by using three different heuristics: the greedy algorithm, the greedy algorithm integrated with limited backtracking, and a general simulated annealing strategy used as a comparison. Figure shows timing on a Pentium 75MHz to stress the fact that the whole technological approach in portable also on widely available machines.

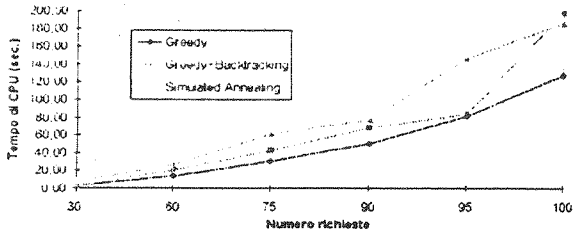


Figure 11: Performances on a Pentium 75MHz

Conclusions

In this paper we have described an approach to the scheduling of the DRS System requests based on a flexible and open architecture named O-OSCAR. The approach from one side relies on efficient algorithms for solution management and from another stresses the need for effective and personalized user interfaces.

The system obtains very interesting performances and solve several limitation of a previously developed rule-based approach (Adinolfi and Cesta 1995).

At present we are increasing the generality of the O-OSCAR framework developing a support for a more general domain description language, inspired by DDL.1 (Cesta and Oddi 1996a), and inserting the ability to deal with multiple capacity resource constraints as described in (Cesta and Stella 1997).

Acknowledgments

This research is supported by ASI — Italian Space Agency and is part of a joint project among Dipartimento di Informatica e Sistemistica dell'Università di Roma "La Sapienza", Dipartimento di Informatica e Automazione della Terza Università di Roma, and Reparto di Intelligenza Artificiale, Modelli Cognitivi ed Interazione at IP-CNR, Roma.

References

M. Adinolfi and A. Cesta. Heuristic Scheduling of the DRS Communication System. *Engineering Applications of Artificial Intelligence*, 8:147–156, 1995.

G. Booch. *Object-Oriented Analysis and Design with Application.* Benjamin Cummings, 1994.

J. Bresina, M. Drummond, and K. Swanson. Expected Solution Quality. In *Proceedings of IJCAI-95*, 1995.

R. Cervoni, A. Cesta, and A. Oddi. Managing Dynamic Temporal Constraint Networks. In *Artificial Intelligence Planning Systems: Proceedings of the Second International Conference (AIPS94)*, 1994.

A. Cesta and A. Oddi. DDL.1: A Formal Description of a Constraint Representation Language for Physical Domains. In M. M.Ghallab and A. Milani, editors, *New Directions in AI Planning.* IOS Press, 1996.

A. Cesta and A. Oddi. Gaining Efficiency and Flexibility in the Simple Temporal Problem. In *Proceedings of the Third International Workshop on Temporal Representation and Reasoning (TIME-96)*, 1996.

A. Cesta and C. Stella. A Time and Resource Problem for Planning Architectures. In *Proceedings of the Fourth European Conference on Planning (ECP 97)*, 1997.

ESA. DRS Preparatory Programme System Performance Specification, Issue 4, Rev. 1. Technical report, European Space Agency, ESTEC, 1989.

ESA. DRS Preparatory Programme Ground Segment Performance Specification, Issue 4, Rev. 1. Technical report, European Space Agency, ESTEC, 1990.

S. French. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop.* Ellis Horwood Lim., 1982.

B. Myers. *The Amulet V.2.0 Reference Manual.* `http://www.cs.cmu.edu/~amulet`.

A. Tate. Mixed-Initiative Interaction in O-Plan. In *Working notes of the AAAI Spring Symposium on Computational Models of Mixed-Initiative Interaction*, 1997.

G. Wolf. Schedule Management: An Object-Oriented Approach. *Decision Support Systems*, 11:373–388, 1994.

Automating Generation of Tracking Plans for a Network of Communications Antennas

S. Chien, A. Govindjee, T. Estlin[a], X. Wang[b], F. Fisher, and R. Hill Jr.[c]
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive, M/S 525-3660
Pasadena, CA 91109-8099
{steve.chien, anita.govindjee, forest.fisher}@jpl.nasa.gov

Abstract

This paper describes the Deep Space Network Antenna Operations Planner (DPLAN) : a system that automatically generates antenna tracking plans for a set of highly sensitive radio science and telecommunications antennas. DPLAN accepts as input an equipment configuration and a set of requested antenna track services. The system then uses a knowledge base of antenna operations procedures to produce a plan of activities that will provide the requested services using the allocated equipment. DPLAN produces this plan using an integration of hierarchical task network (HTN) and operator-based planning. A prototype of the DPLAN system was successfully demonstrated in February 1995 at NASA's experimental DSN station, DSS-13, on a series of Voyager tracks. Based on this successful demonstration, DPLAN is being considered for inclusion in the larger Network Monitor and Control (NMC) upgrade underway projected to save NASA over $9 million per year in operations costs.

Introduction

The Deep Space Network (DSN) [6] was established in 1958 and since has evolved into the largest and most sensitive scientific telecommunications and radio navigation network in the world. The purpose of the DSN is to support unpiloted interplanetary spacecraft missions and to support radio and radar astronomy observations taken in the exploration of the solar system and the universe. There are three deep space communications complexes, located in Canberra, Australia, Madrid, Spain, and Goldstone, California. Each DSN complex operates four deep space stations -- one 70-meter antenna, two 34-meter antennas, and one 26-meter antenna. The function of the DSN is to receive telemetry signals from spacecraft, transmit commands that control spacecraft operating modes, generate the radio navigation data used to locate and guide a spacecraft to its destination, and acquire flight radio science, radio and radar astronomy, very long baseline interferometry (VLBI), and geodynamics measurements.

From its inception the DSN has been driven by the need to create increasingly more sensitive telecommunications devices and better techniques for navigation. Currently, operation of the DSN communications complexes requires a high level of manual interaction with the devices involved in communications links with spacecraft. Recently, NASA has added some new requirements to the development of the DSN: (1) reduce the cost of operating the DSN, (2) improve the operability, reliability, and maintainability of the DSN, and (3) prepare for a new era of space exploration with the New Millennium program, whose goal is to support small, intelligent spacecraft requiring very few mission operations personnel.

This paper describes the Deep Space Network Antenna Operations Planner (DPLAN), which automatically generates plans for individual antenna tracks based on requested services and equipment allocation. The DPLAN system is one element of a far-reaching effort to upgrade and automate DSN operations in order to achieve the three NASA goals mentioned above. A prototype of the DPLAN system was successfully demonstrated in February 1995 at NASA's experimental DSN station, DSS-13 [12,13], on a series of Voyager tracks. Due to this successful demonstration, DPLAN is being considered for inclusion as part of the Network Monitor and Control (NMC) upgrade to DSN stations. The NMC upgrade is projected to enable automation saving NASA over $9 million per year in DSN operations costs.

This paper is organized as follows. We begin by describing the problem of generating Deep Space Network tracking plans. Next, we describe the DPLAN system, including: (1) the track plan generation problem; (2) an overview of artificial intelligence hierarchical task network

---

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

This paper appears in the Working Notes of the 1997 NASA Workshop on Planning and Scheduling for Space.

a - Current Address: Department of Computer Science, University of Texas, Austin, TX, esltin@cs.utexas.edu

b - Current address: Rockwell Science Center, Palo Alto, CA, mei@rpal.rockwell.com

c - Current address: USC/ISI, Marina Del Rey, CA, hill@isi.edu

(HTN) and operator-based planning; (3) the DPLAN system; and (4) an example of operation. Finally, we describe current efforts to deploy the DPLAN system in the operational DSN and other areas of current work.

Track Plan Generation: Problem Description

Each day, at sites around the world, NASA's Deep Space Network (DSN) antennas and subsystems are used to perform scores of tracks that support earth orbiting and deep space missions. Due to the complexity of this equipment, the large set of communications services (in the tens), and the large number of supported equipment configurations (in the hundreds), correctly and efficiently operating this equipment to fulfill tracking goals is a daunting task. An additional requirement is that the antenna operations knowledge embodied in the system must be easily understandable and maintainable. This requirement must also be met as equipment upgrades, services, protocols, and software changes evolve.

The Deep Space Network Antenna Operations Planner (DPLAN) is an automated planning system developed by the Jet Propulsion Laboratory (JPL) to automatically generate antenna tracking plans that satisfy DSN service requests. In order to generate these antenna operations plans, DPLAN uses a number of information sources, including: the project generated service request, the spacecraft sequence of events, the track equipment allocation, and an antenna operations knowledge base.

The project service request represents the basic communications services requested during the track (telemetry/downlink, commanding/uplink, ranging (uplink and downlink), etc.). The project sequence of events indicates the relevant spacecraft mode changes (such as transmission bit rate changes, modulation index changes, etc.). The equipment allocation dictates the antenna and subsystem configuration available for the track. The antenna operations knowledge base provides necessary information on the requirements of antenna operation actions. In particular, this information dictates how these actions can be combined to provide essential communications services.

Artificial Intelligence Planning Techniques

AI planning researchers have developed numerous approaches to the task of correct and efficient planning. Two main planning methods are *operator-based* planning and *hierarchical task network* (HTN) planning. DPLAN uses a combination of both these approaches, exploiting the advantages of each.

Both HTN and operator-based planners typically construct plans by searching through a plan-space. However, they differ considerably in how they search. HTN planners specify plan modifications in terms of flexible task reduction rules and work in a forward-chaining, top-down fashion. In contrast, operator-based planners work in a backward-chaining manner by taking a given goal and attempting to resolve its preconditions. Operator-based planners perform all reasoning at the lowest level of abstraction and provide a strict semantics for defining operator definitions.

An HTN planner [7] uses task reduction rules to decompose abstract goals into lower level tasks. HTN planners can encode many different types of information into task reductions. By defining or not defining certain reduction refinements, the designer can direct the planner towards particular search paths in certain contexts. The user can also directly influence the planner by explicitly adding an ordering constraint or goal protection that would not strictly be derived from goal interaction analyses. Search-control knowledge can also be encoded by writing explicit action sequences to achieve goals, thereby avoiding considerable search.

In contrast, an operator-based planner [15][1] reasons at a single level of abstraction -- the lowest level. Actions are strictly defined in terms of preconditions and effects. Plans are produced through subgoaling and goal interaction analyses. In this framework, all plan constraints (protections, ordering, and codesignation) are a direct consequence of goal achievement and action precondition and effect analysis. Thus, an operator-based planner generally has a strict semantics grounded in explicit state representation, *i.e.* defining what is and is not true in a particular state (or partial state).

The DPLAN planner combines these two planning methods, utilizing the advantages of each. For instance, an operator-based planner requires a very rigid representation which is both a strength and a weakness. It is an advantage in that there is usually one obvious method of encoding each subproblem. However this rigidity can also make certain aspects of a problem difficult to represent. Known ordering constraints and operator sequences can be difficult to encode if they cannot easily be represented in terms of preconditions and effects. Such constraints can and are often forced by adding "dummy" preconditions - in which an operator A is made to precede an operator B by forcing A to achieve a condition C for B. However this solution can often create a misleading representation in that other occurrences of A don't require C to be true. An HTN planner, on the other hand, allows the easy representation of known ordering constraints. Domain information, such as constraints, is easily added to domain rules in the HTN framework. This type of representation allows the user to easily direct the planner's search by explicitly defining items such as ordering constraints and goal protections.

By using a combination of both HTN planning and operator-based planning we can easily direct search and can define knowledge in an understandable top-down fashion. In a hybrid representation we also have the ability to define knowledge in the more structured operator-based fashion when appropriate.

DPLAN's algorithm is a combination of both hierarchical task network (HTN) planning techniques and operator-based

planning techniques. In HTN planning, abstract actions such as "calibrate receiver" or "configure sequential ranging assembly" are decomposed into specific directives for specific hardware types. In operator-based planning, requirements of specific actions, such as "move antenna to point", are satisfied using means-end analysis, which matches action preconditions to effects and resolves any occurring ordering conflicts.

The DPLAN Planning Algorithm

The DPLAN planning algorithm uses a unique combination of the HTN and operator-based planning techniques discussed above. DPLAN operates by refining a set of input top-level goals into a set of low-level operational goals. Plans are represented by a three-tuple: <U,C,S> where U is a set of non-operational (or high-level) goals, C is a set of constraints, and S is a set of operational-goals. At the end of planning, U should be empty and the goals in S are returned as the final plan steps.

A overview of DPLAN algorithm is shown in Figure 1. The main inputs to DPLAN are: a set of high-level goals G, a set of decomposition rules R, and the set of all possible operational goals O. Search is implemented by keeping a queue of partial plans to be explored. Currently, plans are selected from the queue using a best-first heuristic; however, other search techniques could easily be employed. Step 1 and Step 2 of the main loop remove the best plan off the queue, and Step 3 checks if that plan is a solution. If no solution has been found then a new goal is selected for refinement in Step 4. Step 5 chooses a refinement strategy for that goal, and in Step 6, any new plans created through that strategy are inserted into the plan queue.

Algorithm DPLAN(G,R,O)
Initialize the plan queue Q := (<G,{ }, { }>)
While Q is not empty and the resource bound has not been exceeded,

1. Select a promising plan P in Q using heuristics,
2. Remove P from Q
3. If P contains only operational-goals, then check context goals in P. If the context goals are achieved, return P. Otherwise goto 1.
4. Choose a non-operational goal **g** from U.
5. Refine **g**.
6. Insert any new plans generated by refinement into Q.

Figure 1 - The DPLAN Search Algorithm

A plan is considered a solution if two conditions are true. The first is that there are no non-operational goals left to be refined. The second condition is that all context goals have been achieved or are directly achievable in the current plan. Context goals are goals which were needed for applying a decomposition rule, but are supposed to be accomplished by some other part of the plan. If all context goals have been achieved, then the plan is returned as a success.

DPLAN can use several different refinement strategies to handle non-operational goals. There are two main types of goals in DPLAN: *activity-goals* and *state- goals*. *Activity-goals* correspond to operational or non-operational activities and are usually manipulated using HTN planning techniques. Operational activity-goals are considered primitive tasks that can be directly executed. Non-operational activity-goals must be further decomposed into operational ones through HTN reduction rules. *State-goals* correspond to the preconditions and effects of activity-goals, and are achieved through operator-based planning. State-goals that have not yet been achieved are also considered non-operational. Figure 2 shows the procedures used for refining these two types of goals. As soon as a refinement strategy is applied to an activity-goal or state-goal, it is removed from the list of non-operational goals.

If *g* is an Activity-Goal,

1. Decompose: For each decomposition rule r in R which can decompose *g*, apply r to produce a new plan P', If all constraints in P' are consistent, then add P' to Q.
2. Simple Establishment: For each activity-goal *g'* in U that can be unified with *g* simple establish *g* using *g'* and produce a new plan P'. If all constraints in P' are consistent, then add P' to Q.

If *g* is a State-Goal,

1. Step Addition: For each activity-goal effect that can unify with *g*, add that goal to P to produce a new plan P'. If the constraints in P' are consistent, then add P' to Q.
2. Simple Establishment: For each activity-goal *g'* in U that has an effect e that can be unified with *g*, simple establish *g* using e and produce a new plan P'. If all constraints in P' are consistent, then add P' to Q.

Figure 2 - Goal Refinement Strategies

DPLAN can also use additional domain information for more efficient and flexible planning. For instance, a planning problem can specify a list of static context facts. These facts represent operational goals that are always considered to be true. Such goals are easy for DPLAN to verify during planning and can help in pruning off search branches. Other possible inputs include sets of preconditions and effects for operational activities, a set of final goals that must be true in the plan solution, and a set of initial goals that are true at the beginning of planning. This information is not required for standard DPLAN operation, but can be very beneficial during planning.

An Example of DPLAN Representation

As mentioned in the preceding section, DPLAN uses several different types of knowledge to construct a plan. A main component of this knowledge is a set of decomposition

rules. These rules specify how the planner can break down nonoperational activity-goals into lower-level operational goals. A sample rule for performing a telemetry antenna track is shown in Figure 3. This rule defines how the general telemetry operation is broken down into steps. The left-hand side (LHS) of a decomposition rule consists of a set of initial goals, and possibly, a number of other constraints that specify when the rule should be applied. All initial goals and specified constraints must be true in the current plan for the rule to be selected. The initial goals of a rule are the nonoperational goals that the rule "decomposes" into lower-level goals. The rule shown above has only one initial goal that checks if a *telemetry track-goal* is present in the current plan.[1]

The right-hand side (RHS) of a rule contains a set of new goals and constraints over those goals. Once a rule is applied, these new goals replace the LHS initial goals in the current plan. The RHS also contains ordering constraints and protections that specify information about the new goals. An ordering constraint specifies that two goals must be placed in a certain partial order in the final TDN. A protection specifies a causal link that exists between goals. This link explains how the effect of one goal achieves the precondition of another goal. Causal links must always be preserved in order to generate a correct plan. Ordering constraints and protections are added to the current plan and must always be kept consistent during planning. For instance, if an ordering constraint is somehow violated during planning, then the current plan is discarded, and the planner selects another plan from the queue to work on.

Sometimes there may be several different rules that can be used to decompose the same initial goal. For instance, in tracks for 70m antennas, there are several different methods for configuring a receiver depending on the type of receiver being used. To represent these different methods, there are several different rules that can be used to decompose the *perform-receiver-configuration* goal (which was asserted by the telemetry rule in Figure 3).

Conversely, the utilization of goal schemas and operator-based planning techniques allows certain constraint information to be more easily expressed in the domain. Ordering constraints that are due to precondition-effect interactions are directly deduced during planning, instead of having to be explicitly listed by the user. In particular, ordering constraints that apply to very specialized goals, as opposed to very general ones, can be more easily expressed through precondition/effect schemas than through decomposition rules. For more information on the advantages and disadvantages of employing HTN and operator-based planning techniques for this type of domain see [4].

[1] Other possible LHS constraints include additional goal conditions that must be present in the plan, context goals, and codesignation constraints, which check whether two variables can or cannot be unified.

```
(decomprule default-telemetry-track
 lhs
   (initialgoals ((track-goal spacecraft-track telemetry
        ?track-id)))
 rhs
   (newgoals
        ((g1 (perform-antenna-controller- configuration
         ?track-id))
         (g2 (configure-metric-data-assembly ?track-id))
         (g3 (perform-microwave-controller-configuration
              ?track-id))
         (g4 (perform-receiver-configuration ?track-id))
         (g5 (perform-telemetry-configuration ?track-id))
         (g6 (move-antenna-to-point ?track-id))
         (g7 (perform-receiver-calibration ?track-id)))
   constraints
        ((before g1 g6)
         (before g7 g3)
         (before g4 g7))))
```

Figure 3 - Decomposition rule for telemetry track

An Operations Example

In order to begin the planning process, DPLAN is provided with a problem specification that contains several lists of information. Specifically, each problem contains a list of decomposition goals, along with possible lists of initial state predicates, static state predicates, and final state predicates. A sample problem for performing telemetry and ranging with a 70m antenna is shown in Figure 4.

The *init-state* field specifies a list of propositions that are true in the initial state of the planner. For instance, as shown in Figure 4, the exciter drive is assumed to be off prior to when the track is performed. The *static-state* field specifies a list of propositions that are always true during planning (*i.e.* can never be deleted), and is commonly used to list equipment types available to the track. The *decompgoals* field holds the list of nonoperational goals that are to be broken down into lower-level goals through the use of decomposition rules. The *final-state* field is a list of propositions that must be true in the final plan.

A final plan contains a large amount of information, including a list of operational goal names (corresponding to TDN blocks), a list of ordering constraints over those goals, and a list of annotations that describes how the plan was built (*i.e.*, what rules and operations were used). Currently the planner outputs this information in the following way. Three output files are created, a text output file, an annotation file, and a graph-input file. The text output file contains a textual listing of blocks and parameters where blocks are listed in a correct ordering (*i.e.*, blocks do no violate any plan ordering constraints). The annotation file contains a textual list of annotations describing the plan and how it was constructed. The graph-input file contains a list of node names and ordering constraints, which can be used to construct a graphical representation of the plan. See Figure 5 for an example of a plan (or TDN) that was

generated for a problem specification such as that shown in Figure 4.

```
(decompproblem TELEM70
  (init-state ((exciter-drive-off track1)
               (range-mode-off track1)
               (test-translator-off track1)))
  (static-state
     ((CCN-equipment-assignment track1 bstring1)
      (isa bstring1 type-B-telemetry-string)
      (CCN-equipment-assignment track1
          APA-70m)
      (isa APA-70m APA)
      (CCN-equipment-assignment track1 bvr1)
      (isa bvr1 BVR)
      (CCN-equipment-assignment track1 rec1)
      (isa rec1 REC)
      (CCN-equipment-assignment track1 ugc1)
      (isa ugc1 UGC)))
  (decompgoals
      ((perform-pre-cal track1)
       (track-goal spacecraft-track telemetry track1)
       (track-goal spacecraft-track ranging track1)))
  (final-state () ))
```

Figure 4 - Problem specification for a telemetry and ranging track

Application Use And Payoff

The DPLAN system was successfully demonstrated in February 1995 at NASA's experimental DSN station, DSS-13, on a series of Voyager tracks. Based on this successful demonstration, DPLAN is being evaluated for inclusion as part of the larger Network Monitor and Control (NMC) upgrade underway projected to save NASA over $9 million per year in operations costs.

The current DPLAN knowledge base for the planner currently supports the 34-meter (34m) and 70-meter (70m) antenna types at the DSN. All valid types of spacecraft passes for each antenna type are implemented in the knowledge base. Spacecraft passes include the following:

- *telemetry* : Telemetry is a downlink with the spacecraft where information is relayed from the spacecraft to the DSN station on earth.
- *ranging* : Ranging is a method of finding the distance between the spacecraft and the earth which requires both an uplink and a downlink to the spacecraft.
- *commanding*: Commanding is an uplink to the spacecraft where commands are sent from the DSN station to the spacecraft, which instructs the spacecraft to carry out given tasks.
- *VLBI ΔDOR:* VLBI (Very Long Baseline Interferometry) uses quasars --- distant space objects --- in order to determine the location of a spacecraft. A VLBI ΔDOR: (ΔDOR = Differential One-way Ranging) service provides information on the spacecraft's angular position by performing: simultaneous observations from two antenna stations of the spacecraft and a quasar, followed by a second observation of the spacecraft to gather doppler data. This data is then used to determine how to maneuver the spacecraft through space to its destination.
- *VLBI clock sync:* VLBI clock sync gives the instantaneous position of two stations relative to a quasar. This pass is performed in order to determine the rate of change of the clocks at the two DSN stations.
- *radio science:* For radio science, the antenna station is used to gather Radio Frequency (RF) signal information from spacecraft transmissions or natural sources (such as a planet or star).

Not all antenna types perform all types of spacecraft passes. For example, the 34m STD (Standard) antenna is not used for any type of VLBI activity. For each of the antenna types all the types of spacecraft passes that the antenna is used for are covered in the DPLAN knowledge base:

- *34m BWG*: 34-meter Beam Wave Guide. Telemetry, commanding, and ranging.
- *34m STD:* 34-meter Standard. Telemetry, commanding, and ranging.
- *34m HEF:* 34-meter High Efficiency. Telemetry, commanding, ranging, VLBI ΔDOR, and radio science.
- *70m BVR:* 70-meter with Block IV or Block V Receiver. Telemetry, commanding, ranging, VLBI ΔDOR, VLBI clock sync, and radio science.

Generating a plan to make an antenna operational and ready to communicate with a given spacecraft is a complex process - requiring careful coordination of multiple pieces of equipment and subsystems.

The total number of rules in the knowledge base (covering all antenna and track types) is 197: 91 decomposition rules (average of 23 decomposition rules per antenna type) and 106 goal schemas. The knowledge base is modular and easily extended to accommodate new antenna types and new subsystems or equipment types. Also, as changes are made to existing antennas, equipment, and subsystems, the rules can easily be modified. For example, if a new type of antenna controller is added to the 34m-HEF antenna, then a new rule is simply added that configures the new antenna controller. Other rules which use the antenna controller rule do not need to be changed because of the decomposition structure of the knowledge base.

All the plans generated by the planner for the different antenna types and their valid spacecraft passes (including a majority of the multiple combinations of passes) have been verified by the DSN operator experts from all three of the DSN complexes: Goldstone, California (October 1995), Madrid, Spain (January 1996), and Canberra, Australia

(May 1996). For example, the 34m-STD antenna can support telemetry, ranging, and commanding spacecraft passes, and any combination of those three types of passes. DPLAN generated all of the resulting 7 combinations of spacecraft passes (telemetry, ranging, telemetry & ranging, telemetry & commanding, etc.). These passes were then verified on paper by the various operator experts as being correct, executable plans in terms of the ordering of the TDN blocks and the inclusion (or exclusion) of sufficient and necessary TDN blocks.

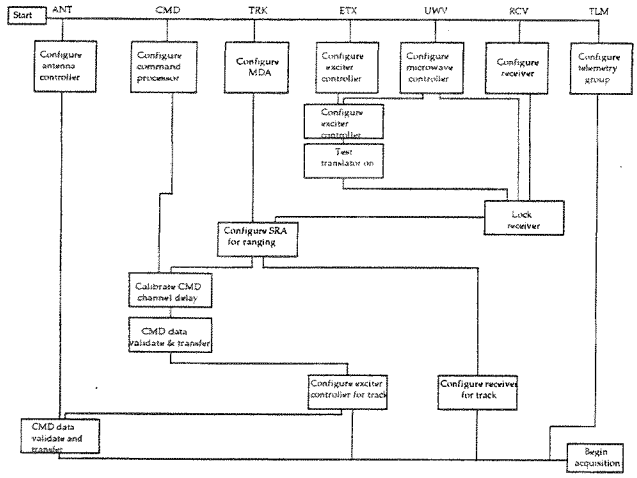


Figure 5 - Temporal Dependency Network for 34 M Beam Wave Guide Antenna Pre-Track for Telemetry, Commanding, and Ranging Services

More testing will occur during the integration phase. During integration, the plans generated by DPLAN are executed by the Automation Engine (AE), which fires scripts associated with each TDN block in the plan. The scripts execute 'operator directives' which turn on and off pieces of equipment, configure subsystems, move the antenna, etc.

A preliminary demonstration was successfully done that integrated the planner with the other elements that comprise the DSN automation. The planner successfully constructed a plan, which was then executed (in simulation) by the AE. This demonstration took place in December 1995. Further testing of the planner took place in August 1996 in a computer simulated antenna environment with simulated subsystems and equipment. DPLAN is currently being considered for inclusion into the NMC's AE for the D1 delivery which is scheduled to occur in August 1997. When fully integrated, the AE will call the planner to generate a given plan and then execute that plan, firing off the necessary scripts for the TDN blocks.

Maintenance

One key issue for DSN antenna operations is maintenance of the software systems necessary to automate DSN antenna operations. It was our experience that DSN operations personnel were quite confident that antenna operations could be automated - the key point was reducing the costs of maintaining expert knowledge regarding operations. In our development of the extensive DSN antenna operations knowledge base, we have collaborated with DSN antenna operation experts (operators and engineers) and it is expected that they will maintain and extend the antenna operations knowledge base. In this section we discuss several issues relevant to maintenance of the DSN antenna operations planner knowledge base including: representation for maintainability and comparisons to other automation representations considered by the DSN.

Representation for Maintainability

An important aspect of the DPLAN representation is that it allows for natural encoding of abstract objects and procedures (*e.g.* receiver calibration). By allowing decomposition rules to refer to abstract objects, changes to DSN procedures involve fewer knowledge base updates than if the knowledge base contained a large number of very specific rules. For instance, a change relating to a specific equipment type need not affect more general domain information. If a new receiver type called a BLOCK-VI receiver were added to the DSN equipment list, more general rules, such as the telemetry rule shown in Figure 3, would not need to be modified. Instead, only a few more specific rules need be constructed or edited. In this case, a new *configure-receiver* rule would be added. Therefore, many such changes would cause only a few specialized rules to be created or updated instead of causing numerous rules to be modified. Even with the current DSN goal to automate all TDN generation, the planning knowledge base must be constantly updated and verified. Fewer more general rules are cheaper to update and verify, and can thus support more efficient knowledge base maintenance.

Another benefit to this type of representation is that domain information is more easily understood. By keeping domain details separate from more general knowledge, it is easier for a user to understand the general aspects of an antenna track. For example, to understand the general steps of a telemetry operation, a user only has to view the main telemetry track decomposition rule. If more low-level knowledge is desired, such as how to operate a particular piece of equipment, the user could then search for rules that directly pertain to that equipment type.

Comparison to Scripts

One option considered by DSN personnel was to implement the higher level of track automation by a hierarchy of scripts. There would be scripts for general activities, such

as calibrating a Block V Receiver in the context of a ranging track. This scripting approach can be viewed as similar to the HTN planning approach, but with two key differences. First, there is no explicit representation of the context in which a script will necessarily achieve the goal. The set of situations in which a script S is expected to work is represented only implicitly in the set of scripts which call S. The intended coverage, conditions, etc. are not explicitly represented as they are in HTN rules. The second difference is that the planner allows a "call by goal" usage in operator-based planning. In this way, the planner can invoke routines (or operators) based on the conditions it desires to achieve, and the planner will automatically detect and resolve any conflicting interactions with other activities. Not only does the planner representation allow for encoding of conditions and assumptions of when particular activities are appropriate (through conditions on HTN rules or preconditions on operators), it actually requires such definitions in order to operate correctly. Therefore it encourages correct documentation of operations requirements for all activities - which should lead to more maintainable operations procedures.

Comparison to End-to-End TDNs

Another option considered by DSN Operations was to simply encode end-to-end TDNs for each supported combination of the cross product between service requests and equipment allocation. Unfortunately, this option has several drawbacks. First, articulating all of the relevant knowledge in this format can be very tedious and prone to error. While generating the initial set of end-to-end TDNs, the expert operators said that they often found it difficult to keep all of the different TDNs straight. Second, this representation is not amenable to maintenance. If an equipment type is added or changed, it must be changed in every TDN that is relevant. The knowledge pertaining to the equipment type is not centralized in a set of rules or activity definitions as it is in the planning representation.

Discussion

In this section we discuss several issues relevant to the DSN planner including representing and reasoning about plan quality and replanning.

Representing and Reasoning about Plan Quality

Representing and reasoning about plan quality [11][16][18] is another key concern of DSN operations. Because of space constraints we only describe the most important aspects of reasoning about plan quality in the DSN antenna operations application - for further details the reader is referred to [3]. Since there is often more than one correct plan for a particular antenna operation, it is important for a planning system to be able to compare a set of final plans using user identified plan quality measures. There are a number of quality measures that can be emphasized during planning, including producing more robust, flexible and/or efficient plans. One important quality goal is to minimize the overall plan execution time. In particular, the time to setup (pre-calibration) and reset (post-calibration) the communications link can often be reduced. For instance, it can take up to two hours to manually pre-calibrate a DSN 70-meter antenna communications link for certain types of mission. By using a plan generated by DPLAN, this time can be reduced to approximately 30 minutes, where further reductions in set-up time are limited by physical constraints of the subsystems themselves.

Plan execution time is often significantly reduced by exploiting parallel path possibilities, especially where the control of multiple subsystems is involved. DPLAN currently uses the critical path length of a plan to help identify better plans. Critical path length is calculated using time information attached to a TDN block, which specifies the average time it should take to execute the block. By comparing critical path lengths of competing plans, DPLAN could choose a highly efficient final plan that will provide a minimal execution time. Minimizing plan execution time allows more data to be returned per operating time for the link.

Another issue for plan quality is plan flexibility. There certain standard TDN blocks that may be inserted into a plan at various points (such as transmission rate changes, etc.). If such commands are executed in the middle of an inflexible plan, it may not be possible to continue execution. Depending on the steps inserted, preconditions, postconditions, and time tags of other blocks may become invalid. Flexible plans that allow for the insertion of common steps while still retaining their applicability are greatly valued.

Replanning for Antenna Tracks

Additionally, DPLAN is required to replan during the course of typical antenna operations. Replanning occurs in two general cases. First, after a plan has been generated, the objectives sometimes change. Often, shortly prior to or during a track, a project may submit a request to add services to the track. This request corresponds to additional goals that must be incorporated into the track plan. In the case where goals are added before the track actually begins, DPLAN adds these unachieved goals to the current plan and restarts the planning process. Unfortunately, this method is incomplete in theory because the planner may have previously made choices that are incompatible with the new goals. However, for the specific sets of goals and domain theories (related to antenna operations) that we have examined, we have been able to use encodings in which completeness has not been a problem. This is an area of current work. Another area of current work is replanning in the case where goals are added during actual track execution. One approach to dealing with this would be to allow the planner the ability to backtrack or repair the current plan so as to adapt to the current situation. The

planner might do this using a set of plan modification operators.

Another replanning issue is caused by dynamism. After a plan has been generated, a block (plan step) may fail, a piece of equipment may require resetting (due to general unreliability), or a piece of equipment may fail or be pre-empted by a higher priority track. In the case of a simple plan step failure, DPLAN simply calls for re-execution of the block. If a piece of equipment requires resetting, DPLAN has knowledge describing which achieved goals have been undone and require re-establishment. DPLAN then uses a replanning technique [20] that re-uses parts of the original plan to re-achieve the undone goals as necessary. This technique takes advantage of the fact that the original plan begins from a state that is equivalent to resetting all of the subsystems.

Conclusions

This paper has described the DSN Antenna Operations Planner (DPLAN) which automatically generates communications antenna tracking plans based on requested services and equipment allocation. DPLAN uses a knowledge base of information on tracking activities and a combination of artificial intelligence planning methods to generate appropriate tracking plans. We have also described the deployment status of the DPLAN system and outlined areas of current work including: representation and reasoning about plan quality, replanning, and representation to support maintainability. The DPLAN system was successfully demonstrated in February 1995 at NASA's experimental DSN station, DSS-13, on a series of Voyager tracks. Based on this successful demonstration, DPLAN is being evaluated for inclusion as part of the larger Network Monitor and Control (NMC) upgrade underway projected to save NASA over $9 million per year in operations costs.

References

[1] J.G. Carbonell, J. Blythe, O. Etzioni, Y. Gil, R. Joseph, D. Kahn, C. Knoblock, S. Minton, M.A. Perez, S. Reilly, M. Veloso, X. Wang, "Prodigy 4.0: The Manual and Tutorial," *Technical Report*, School of Computer Science, Carnegie Mellon University.

[2] S. A. Chien, R. W. Hill Jr., X. Wang, T. Estlin, K. V. Fayyad, and H. B. Mortensen, "Why Real-world Planning is Difficult: A Tale of Two Applications," in *New Directions in AI Planning*, M. Ghallab and A. Milani, ed., IOS Press, Washington, DC, 1996, pp. 287-298.

[3] S. Chien, A. Govindjee, X. Wang, T. Estlin, R. Hill, Jr., "Automated Generation of Tracking Plans for a Network of Communications Antennas," *Proceedings of the 1997 IEEE Aerospace Conference*, Aspen, CO, February 1997.

[4] T. Estlin, S. Chien, , and X. Wang, "An Argument for an Integrated Hierarchical Task Network and Operator-based Approach to Planning," *Proceedings of the 1997 European Conference on Planning*, Toulouse, France, September 1997, Lecture Notes in Artificial Intelligence, Springer—Verlag.

[5] S. Chien, R. Lam, and Q. Vu, "Resource Scheduling for a Network of Communications Antennas," *Proceedings of the 1997 IEEE Aerospace Conference*, Aspen, CO, February 1997.

[6] Deep Space Network, *Jet Propulsion Laboratory Publication 400-517*, April 1994.

[7] K. Erol, J. Hendler, and D. Nau, "UMCP: A Sound and Complete Procedure for Hierarchical Task Network Planning," *Proceedings of the Second International Conference on AI Planning Systems*, Chicago, IL, June 1994, pp. 249-254.

[8] Fayyad, K. E. and L. P. Cooper. "Representing Operations Procedures Using Temporal Dependency Networks," *SpaceOps '92*, Pasadena, CA, November 1992.

[9] K. Fayyad, R.W. Hill, Jr., and E.J. Wyatt. "Knowledge Engineering for Temporal Dependency Networks as Operations Procedures." *Proc. AIAA Computing in Aerospace 9 Conf.* , 1993, San Diego, CA.

[10] R. W. Hill, Jr., S. A. Chien, K. V. Fayyad, C. Smyth, T. Santos, and R. Bevan, "Sequence of Events Driven Automation of the Deep Space Network," *Telecommunications and Data Acquisition* 42-124, October-December 1995.

[11] J. S. Pemberthy and D. S. Weld, "UCPOP: A Sound Complete, Partial Order Planner for ADL," *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*, October 1992.

[12] M. Williamson and S. Hanks, "Optimal Planning with a Goal-directed Utility Model," *Proceedings of the Second International Conference on AI Planning Systems*, Chicago, IL, June 1994, pp. 176-180.

[13] A. Perez and J. Carbonell, "Control Knowledge to Improve Plan Quality," *Proceedings of the Second International Conference on AI Planning Systems*, Chicago, IL, June 1994, pp. 323-328.

[14] Final Report of the Services Fulfillment Reengineering Team, *JPL Interoffice Memorandum*, March 14, 1995.

[15] X. Wang and S. Chien, "Replanning while using both Hierarchical Task Network and Operator-based Planning," *Proceedings of the 1997 European Conference on Planning*, Toulouse, France, September 1997, Lecture Notes in Artificial Intelligence, Springer—Verlag.

Using Artificial Intelligence Planning to Automate Science Data Analysis for Large Image Databases

Steve Chien, Forest Fisher, Helen Mortensen
Jet Propulsion Laboratory
California Institute of Technology

Edisanter Lo, Ronald Greeley
Department of Geology
Arizona State University, Tempe, AZ

Abstract

In recent times, improvements in imaging technology have made available an incredible array of information in image format. While powerful and sophisticated image processing software tools are available to prepare and analyze the data, these tools are complex and cumbersome, requiring significant expertise to properly operate. Thus, in order to extract (e.g., mine or analyze) useful information from the data, a user (in our case a scientist) often must possess both significant science and image processing expertise.

This paper describes the use of AI planning techniques to represent scientific, image processing, and software tool knowledge to automate knowledge discovery and data mining (e.g., science data analysis) of large image databases. In particular, we describe two fielded systems. The Multimission VICAR Planner (MVP) which has been deployed for 2 years and is currently supporting science product generation for the Galileo mission. MVP has reduced time to fill certain classes of requests from 4 hours to 15 minutes. The Automated SAR Image Processing system (ASIP) which is currently in use by the Dept. of Geology at ASU supporting aeolian science analysis of synthetic aperture radar images. ASIP reduces the number of manual inputs in science product generation by 10-fold.

Introduction

Recent breakthroughs in imaging technology have led to an explosion of available data in image format. However, these advances in imaging technology have brought with them a commensurate increase in the complexity of image processing and analysis technology. When analyzing newly available image data to discover patterns or to confirm scientific theories, a complex set of operations is often required. First, before the data can be used it must often be reformatted, cleaned, and many correction steps must be applied. Then, in order to perform the actual data analysis, the user must manage all of the analysis software packages and their requirements on format, required information, etc.

Furthermore, this data analysis process is not a one-shot process. Typically a scientist will set up some sort of analysis, study the results, and then use the results of this analysis to modify the analysis to improve it. This analysis and refinement cycle may occur many times - thus any reduction in the scientist effort or cycle time can dramatically improve scientist productivity.

Unfortunately, this data preparation and analysis process is both knowledge and labor intensive. Consider the task of producing a mosaic of images of the moon from the Galileo mission (corrected for lighting, transmission errors, and camera distortions). Consider also that our end goal is to perform geological analyses - i.e., to study the composition of the surface materials on the moon. One technique used to do this is to construct a ratio image - an image whose values are the ratio of the intensity of the response at two different bandwidths (e.g., the ratio of infra-red response and visible green response). In order to correctly be able to produce this science product for analysis, requires knowledge of a wide range of sources including:

- the particular science discipline of interest (e.g., atmospheric science, planetary geology),
- image processing and the image processing libraries available,
- where and how the images and associated information are stored (e.g., calibration files), and
- the overall image processing environment to know how to link together libraries and pass information from one program to another.

It takes many years of training and experience to acquire the knowledge necessary to perform these analyses. Needless to say, these experts are in high demand.

[0]This paper describes research conducted by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. The authors would also like to acknowledge the contributions of other past and present members of the MVP team: Todd Turco, Christine Ying, Shouyi Hsiao, Darren Mutz, Alex Gray, Joe Nieten, and Jean Lorre. The authors would also like to acknowledge other contributors to the ASIP project including: Dan Blumberg(ASU), Anita Govindjee, John McHone(ASU), Keld Rasmussen(ASU), and Todd Turco.

One factor which exacerbates this shortage of experts is the extreme breadth of knowledge required. Many users might be knowledgable in one or more of the above areas but not in all the areas. In addition, the status quo requires that users possess considerable knowledge about software infrastructure. Users must know how to specify input parameters (format, type, and options) for each software package that they are using and must often expend considerable effort in translating information from one package to another.

Using automated planning technology to represent and automate many of these data analysis functions (p. 50 (Fayyad96)) enables novice users to utilize the software libraries to mine the data. It also allows users who may be expert in some areas but less knowledgable in other to use the software tools to mine the data.

The remainder of this article is organized as follows. First, we provide a brief overview of the key elements of AI planning. We then describe two fielded planning systems for science data analysis. We first describe the MVP system - which automates elements of image processing for science data analysis for data from the Galileo mission. We then describe the ASIP system - which automates elements of image processing for science data analysis of synthetic aperture radar (SAR) images.

The principle contributions of this article are twofold. First, we identify software tool reconfiguration as an area where AI planning technology can significantly extend KDD capabilities. Second, we describe two systems demonstrating the viability and impact of AI planning on the KDD process.

Artificial Intelligence Planning Techniques

We have applied and extended techniques from Artificial Intelligence Planning to address the knowledge-based software reconfiguration problem in general, and two applications in science data analysis (e.g., data mining) in specific. In order to describe this work, we first provide a brief overview of the key concepts from planning technology [1].

Planning technology relies on an encoding of possible actions in the domain. In this encoding, one specifies for each action in the domain: *preconditions*, *postconditions*, and *subactivities*. Preconditions are requirements which must be met before the action can be taken. These may be pieces of information which are required to correctly apply a software package (such as the image format, availability of calibration data, etc. Postconditions are things that are made true by the execution of the actions, such as the fact that the data has been photometrically corrected (corrected for the relative location of the lighting source) or that 3-dimensional topography information has been extracted from an image. Substeps are lower level activities which comprise the higher level activity. Given this encoding of actions, a planner is able to solve individual problems, where each problem is a current state and a set of goals. The planner uses its action models to synthesize a plan (a set of actions) to achieve the goals from the current state.

Planning consists of three main mechanisms: subgoaling, task decomposition, and conflict analysis. In subgoaling, a planner ensures that all of the preconditions of actions in the plan are met. This can be done by ensuring that they are true in the initial state or by adding appropriate actions to the plan. In task decomposition, the planner ensures that all high level (abstract) activities are expanded so that the lower level (subactivities) are present in the plan. This ensures that the plan consists of executable activities. Conflict analysis ensures that different portions of the plan do not interfere with eachother.

The Multimission VICAR Planner (MVP)

MVP (Chien96) partially automates generation of image processing procedures from user requests and a knowledge-based model of VICAR image processing area using Artificial Intelligence (AI) automated planning techniques. In VICAR image processing, the actions are VICAR image processing programs, the current state is the current state of the image files of interest, and the specification of the desired state corresponds to the user image processing goals.

The VICAR environment (Video Image Communication and Retrieval [2]) (LaVoie89) supports image processing for: JPL flight projects including VOYAGER, MAGELLAN, and GALILEO, and CASSINI; other space imaging missions such as SIR-C and LANDSAT; and numerous other applications including astronomy, earth resources, land use, biomedicine, and forensics with a total of over 100 users. VICAR allows individual processing steps (programs) to be combined into more complex image processing scripts called procedure definition files (PDFs). The primary purpose of VICAR is to enable PDFs for science analysis of image data from JPL missions.

An Example of MVP Usage

In order to illustrate how MVP assists in VICAR planetary image processing, we now provide a typical example of MVP usage to ground the problem and the inputs and outputs required by MVP. The three images, shown at the left of Figure 1 are of the planet Earth taken during the Galileo Earth 2 flyby in December 1992. However, many corrections and processing steps must be applied before the images can be used. First, errors in the compression and transmission of the data from the Galileo spacecraft to receivers on Earth has resulted in missing and noisy lines in the images. Line fillin and spike removals are therefore desirable. Second, the images

[1] For Further details on planning the user is referred to (Pemberthy92; Erol94)

[2] This name is somewhat misleading as VICAR is used to process considerable non-video image data such as MAGELLAN synthetic aperture radar (SAR) data.

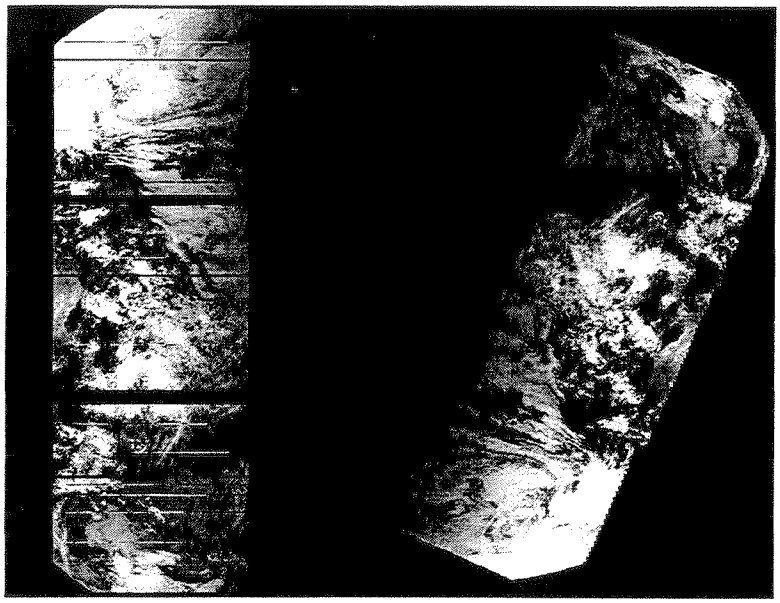


Figure 1: Raw and Processed Image Files

should be map projected to correct for the spatial distortion that occurs when a spherical body is represented on a flat surface. Third, in order to combine the images, we need to compute common points between the images and overlay them appropriately. Fourth, because we are combining multiple images taken with different camera states, the images should be radiometrically corrected before combination.

MVP enables the user to input image processing goals through a graphical user interface with most goals as toggle buttons on the interface. A few options require entering some text, usually function parameters that will be included as literals in the appropriate place in the generated VICAR script. Figure 2 shows the processing goals input to MVP. Using the image processing goals and its

radiometric correction	pixel spike removal
missing line fillin	uneven bit weight correction
no limbs present in images	perform automatic navigation
display automatic nav residual error	perform manual navigation
display man nav residual error	map project with parameters ...
mosaic images	smooth mosaic seams using DN

Figure 2: Example Problem Goals

knowledge of image processing procedures, MVP constructs a plan of image processing steps to achieve the requested goal. This plan is translated into a VICAR script which, when run, performs the desired image corrections and constructs a mosaicked image of the three input files. The finished result of the image processing task is shown at the right in Figure 1. The three original images now appear as a single mosaicked image, map projected with missing and corrupted lines filled in.

To further continue this example, shown in Figure 3 is a code fragment to perform portions of image navigation

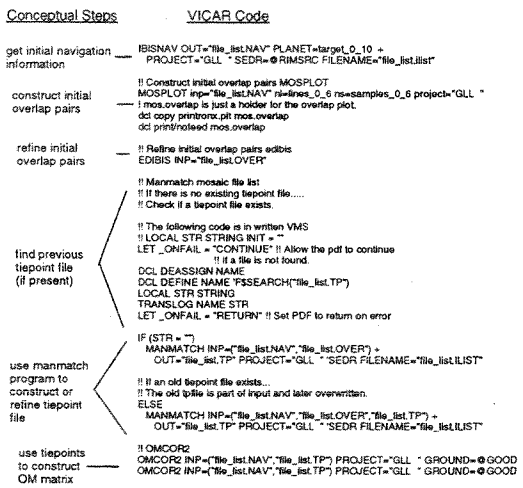


Figure 3: Sample VICAR Code Fragment

[3] for a Galileo image [4]. The higher-level conceptual steps (i.e., plan steps) are shown at the left and the corresponding VICAR code is shown at the right. In this case the overall user goal is to navigate the image. The other subgoals (and steps) are necessary to support this goal.

Thus MVP allows the user to go directly from high

[3] Image navigation is the process of determining the matrix transformation to map from the 2-dimensional (line, sample) coordinate space of an image to a 3-dimensional coordinate space using information on the relative position of the imaging device (spacecraft position) and a model of the target being imaged (e.g., the planetary body).

[4] This code was generated by MVP.

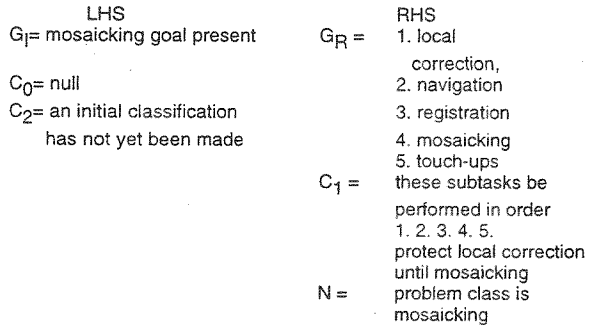


Figure 4: A Task Reduction Rule from MVP

level image processing goals to an executable image processing program. By insulating the user from many of the details of image processing, productivity is enhanced. The user can consider more directly the processing goals relevant to the end science analysis of the image, rather than being bogged down in the details such as file format, normalizing images, etc.

MVP does not always fully automate this planetary imaging task. In typical usage, the analyst receives a request, determines which goals are required to fill the request, and runs MVP to generate a VICAR script. The analyst then runs this script and then visually inspects the produced image(s) to verify that the script has properly satisfied the request. In most cases, upon inspection, the analyst determines that some parameters need to be modified subjectively or goals reconsidered in context. This process typically continues several iterations until the analyst is satisfied with the image product.

Task Reduction in MVP

MVP represents VICAR processing and science data analysis knowledge in the form of task reduction rules. For example, Figure 4 shows a decomposition rule for the problem class *mosaicking with absolute navigation*. This rule states that if mosaicking is a goal of the problem and an initial problem decomposition has not yet been made, then the initial problem decomposition should be into the subproblems local correction, navigation, registration, mosaicking, and touch-ups and that these subproblems must be solved in that order.

Operator-based Planning in MVP

MVP also uses operator-based planning techniques (Pemberthy92). An operator-based planner uses models of actions in a domain to achieve goals from an initial world state. In the VICAR domain the actions (operators) are image processing steps, initial state the initial image file state, and the goals the processing request.

In operator-based planning, an action is represented in terms of its preconditions (required to be true before an action can be executed), and its effects (true after an action is executed). For example, the GALSOS program to radiometrically correct Galileo image files is represented as shown below.

```
operator GALSOS
        :parameters ?infile ?ubwc ?calc
        :preconditions the project of ?infile must be galileo
                       the data in ?infile must be raw data values
        :effects       reseaus are not intact for ?infile
                       the data in ?infile is not raw data values
                       missing lines are not filled in for ?infile
                       ?infile is radiometrically corrected
                       the image format for ?infile is halfword
                       ?infile has blemishes-removed
                       if (UBWC option is selected)
                       then ?infile is uneven bit weight corrected
                       if (CALC option is selected)
                       then ?infile has entropy values calculated
```

When constructing a plan to achieve a goal G1, a planner will consider those actions which have G1 as an effect (thus considering GALSOS to achieve a radiometric correction goal). In order to use GALSOS, MVP must also ensure that the preconditions of the operator are met, in a process called *subgoaling*. MVP must also ensure that operators in the plan do not undo conditions required by other parts of the plan - this is performed in a process called *conflict analysis* [5].

One novel aspect of the VICAR domain is that considerable search in planning is not at the *program selection level* (which corresponds to operator selection in the planning process) but rather at the *program option selection level* (which corresponds to selecting the appropriate operator effect after the operator has been selected). In order to efficiently handle this type of search, we have integrated a constraint reasoning mechanism which allows the planner to reason about compatible and incompatible program option settings in a least-commitment fashion (see (Chien96) for details).

An Example of Subgoaling in VICAR Image Processing To illustrate how the operator-based planning process performs subgoaling, consider the subgoal graph illustrated in Figure 5.[6] In this case the user has selected the goal that the images be navigated using manual methods and that the archival navigation information for the image should be updated. The decomposition planner has access to the knowledge that in order to navigate the image, the operational goal is to construct an OM matrix which defines the transformation from (line, sample) in the image to some known frame of reference (usually the position relative to the target planet center). The planner knows that in order to compute this matrix it must have a tiepoint file, the project of the image, and the image files formatted into a mosaic file list. In order to produce a tiepoint file for the goal specification of manual navigation, the planner uses the MANMATCH program. The MANMATCH program in turn requires a refined overlap pairs file, the project of the images, the initial predict information, and again a mosaic file list.

[5] Because subgoaling and conflict analysis in operator-based planning are not unique to MVP, we have only briefly sketched their key elements. For a more detailed treatment of operator-based planning algorithms the reader is referred to (Pemberthy92).

[6] The VICAR code previously shown in Figure 3 is taken from this example.

The refined overlap pairs file can be constructed using the EDIBIS program, but this requires a crude overlap pairs file based on an initial predict source. This crude overlap pairs file in turn requires the default navigation method, and the latitude and longitude of sample image files. The rest of the graph is generated similarly. This subgoal graph is generated in response to the particular combination of user goals and the state of the selected image files.

An Example of Resolution of Goal Conflicts in VICAR Image Processing To illustrate how the operator-based planning process resolves interactions between steps, consider the (simplified) image processing operators shown in Figure 6. The relevant operators to achieve the goals of missing line fillin, spike removal, and radiometric correction for Voyager and Galileo images are shown below. When constructing a plan to achieve these goals, depending on the project of the image file (e.g., either Voyager or Galileo), MVP determines the correct program to use because the preconditions enforce the correct program selection.

However, determining the correct ordering of actions can sometimes be complex. In this case, the correct order to achieve the goals of line fillin, spike removal, and radiometric correction is dependent upon the project of the file. In the case of Voyager files, ADESPIKE (spike removal) requires raw pixel values and FICOR77 (radiometric correction) changes pixel values to correct for camera response function – thus FICOR77 removes a necessary condition for ADESPIKE (raw pixel values). This interaction can be avoided by enforcing that ADESPIKE occurs before FICOR77. Additionally, VGRFILLIN requires binary EDR header on the image file, and ADESPIKE removes the binary EDR header, thus ADESPIKE removes a necessary condition for VGRFILLIN. This interaction can be avoided by requiring VGRFILLIN to be executed before ADESPIKE. Thus in the VOYAGER example the only legal execution order is VGRFILLIN, ADESPIKE, FICOR77.

In the Galileo case, GALSOS undoes missing line fillin (the goal achieved by the GLLFILLIN operator). Thus in order to avoid undoing this processing, GLLFILLIN must be applied after GALSOS. Additionally, GALSOS requires raw pixel values, and ADESPIKE alters the pixel values, so ADESPIKE removes a necessary condition for GALSOS. This interaction can be avoided by requiring that GALSOS occurs before ADESPIKE. This determination of correct programs to use and correct execution order is shown in Figure 7.

This simple example illustrates some of the interactions and context-sensitivity of the VICAR image processing application. All of these interactions and context sensitive requirements are derived and accounted for automatically by MVP using the operator specification, thereby allowing plan construction despite the presence of complex interactions and conditions.

Impact of the MVP system

User reports indicate that MVP reduces effort to generate an initial PDF for an expert analyst from 1/2 a day to 15 minutes and reduces the effort for a novice analyst from several days to 1 hour. This represents over an order of magnitude in speedup. The analysts also judged that the quality of the PDFs produced using MVP are comparable to the quality of completely manually derived PDFs.

The Automated SAR Image Processing (ASIP) System

ASIP automates synthetic aperture radar (SAR) image processing based on user request and a knowledge-base model of SAR image processing using AI automated planning techniques. SAR operates simultaneously in multipolarizations and multifrequencies to produce different images consisting of radar backscatter coefficients (s0) through different polarizations at different frequencies. ASIP enables construction of an aerodynamic roughness image/map (z0 map) from this raw data - thus enabling studies of Aeolian processes.

Studies of Aeolian Processes

The aerodynamic roughness length (z0) is the height above a surface at which a wind profile assumes zero velocity. z0 is an important parameter in studies of atmospheric circulation and aeolian sediment transport (in laymans terms: wind patterns, wind erosion patterns, and sand/soil drift caused by wind) (Greeley87a; Greeley87b; Greeley91). Estimating z0 with radar is very beneficial because then large areas can be mapped quickly to study aeolian processes, as opposed to the slow painstaking process of manually taking field measurements(Blumberg95). The final science product is a VICAR image called a z0 map that the scientists use to study the aeolian processes.

Planning to Generate Aerodynamic Roughness Maps

ASIP is an end-to-end image processing system automating data abstraction, decompression, and (radar) image processing sub-systems, and intelligently integrates a number of SAR and z0 image processing subsystems. Using a knowledge base of SAR processing actions and a general-purpose planning engine, ASIP reasons about the parameter and sub-system constraints and requirements. In this fashion ASIP extracts needed parameters from image format and header files as appropriate, relieving the user of having to know about these aspects of the problem. These parameters, in conjunction with the knowledge-base of SAR processing steps, and a minimal set of required user inputs (entered through a single graphical user interface (GUI)), are then used to create the processing plan. ASIP represents a number of processing constraints. For example, ASIP represents the fact that only some subset of all

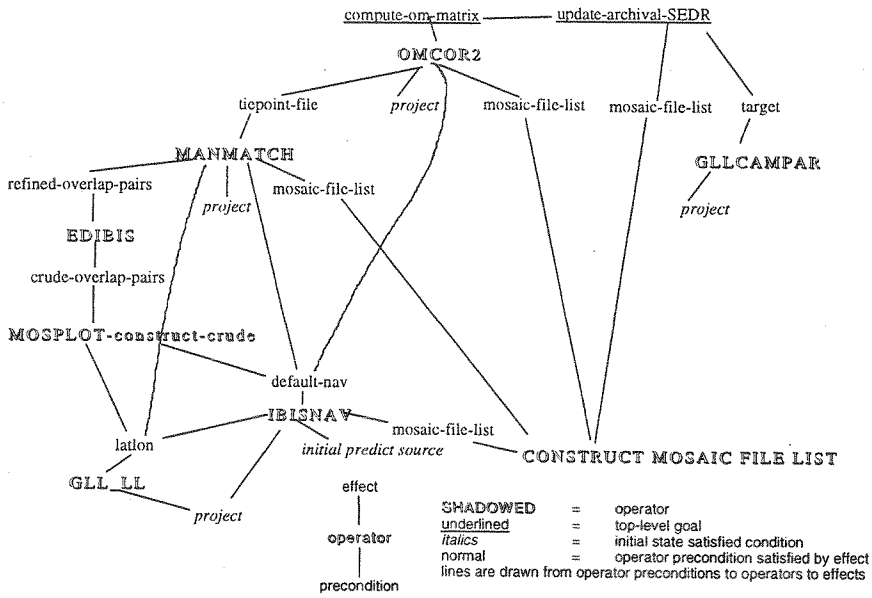


Figure 5: Subgoal Graph for Manual Relative Navigation of Galileo Image Files

Operator	VGRFILLIN	GLLFILLIN	ADESPIKE	FICOR77	GALSOS
Preconditions	VGR image EDR present	GLL image	(GLL image) or ((VGR image) and (raw values))	VGR image	GLL image raw pixel values
Effects	missing lines filled in.....		spike removal not raw values	radiometric corr. blemish removal not raw values	radiometric corr. reed-solomon overflow corr. saturated pixel corr. not missing line fillin

Figure 6: Simplified Operator Definitions

	VICAR Program		Execution Order	
Goal	Voyager	Galileo	Voyager	Galileo
fillin missing lines	VGRFILLIN	GLLFILLIN	VGRFILLIN	GALSOS
remove spikes	ADESPIKE	ADESPIKE	ADESPIKE	GLLFILLIN
radiometric corr.	FICOR77	GALSOS	FICOR77	ADESPIKE

Figure 7: VICAR Programs and Execution Order to Resolve Conflicts for Voyager and Gallieo Data

possible combinations of polarizations are legal (as dependent on the input data). ASIP also represents image processing knowledge about how to use polarization and frequency band information to compute parameters used for later processing of backscatter to aerodynamic roughness length conversion - thus freeing the user from having to understand these processes.

Figure 8 shows an aerodynamic roughness length map of a site near Death Valley, California generated using the ASIP system (the map uses the L band (24 cm) SAR with HV polarization). Each of the greyscale bands indicated signifies a different approximate aerodynamic roughness length. This map is then used to study aeolian processes at the Death Valley site.

Since the ASIP system has been fielded, it has proven to be very useful in the use of generating aerodynamic roughness maps with three major benefits. First, ASIP has enabled a 10 fold reduction in the number of manual inputs required to produce an aerodynamic roughness map. Second, ASIP has enabled a 30% reduction in CPU processing time to produce such a map. Third, and most significantly ASIP has enabled scientists to process their own data (previously programming staff were required). By enabling scientists to directly manipulate that data and reducing processing overhead and turnaround, science is directly enhanced.

Conclusions

This paper has described knowledge-based reconfiguration of data analysis software using AI planning techniques. This represents an important area where AI planning can significantly enhance KDD processes. As evidence of this potential, we described two fielded planning systems that enhance KDD: the MVP system, which automates image processing to support Galileo image data science analysis; amd the ASIP system which automates production of aerodynamic roughness maps to support geological science analysis.

References

D. Blumberg and R. Greeley. "Field Studies of Aerodynamic Roughness Length." Journal of Arid Environments (1993) 25:39-48.

S. A. Chien and H. B. Mortensen, "Automating Image Processing for Scientific Data Analysis of a Large Image Database," IEEE Transactions on Pattern Analysis and Machine Intelligence 18 (8): pp. 854-859, August 1996.

K. Erol, J. Hendler, and D. Nau, "UMCP: A Sound and Complete Procedure for Hierarchical Task Network Planning," Proceedings of the 2nd International Conference on AI Planning Systems, Chicago, IL, June 1994, pp. 249-254.

U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, "From Data Mining to Knowledge Discovery in Databases," AI MAgazine, Vol 17 No. 3, Fall 1996, pp. 37-54.

F. Fisher, E. Lo, S. Chien, R. Greeley, "Automated SAR Processing for Science Analysis using Artificial Intelligence Planning Techniques," Submitted to the 1997 International Conference on Image Processing, Santa Barbara, CA, November 1997.

Y. Iwasaki and P. Friedland, "The Concept and Implementation of Skeletal Plans,"Journal of Automated Reasoning 1, 1 (1985), pp. 161-208.

R. Greeley and J.D. Iversen, "Measurements of Wind Friction Speeds over Lava Surfaces and Assessment of Sediment Transport."G.R.L. 14 (1987):925-928.

R. Greeley, P.R. Christensen, and J.F. McHone. "Radar Characteristics of Small Craters: Implications for Venus." EMP 37 (1987):89-111.

R. Greeley, L. Gaddis, A. Dobrovolskis, J. Iversen, K. Rasmussen, S. Saunders, J. vanZyl, S. Wall, H. Zebker, and B. White. "Assessment of Aerodynamic Roughness Via Airborne Radar Observations." 1991, Acta Mechanica Suppl.2, 77-88.

A. Lansky, M. Friedman, L. Getoor, S. Schmidler, and N. Short Jr., "The Collage/Khoros Link: Planning for Image Processing Tasks," Proceedings of the 1995 AAAI Spring Symposium on Integrated Planning Applications, March 1995, pp. 67-76.

S. LaVoie, D. Alexander, C. Avis, H. Mortensen, C. Stanley, and L. Wainio, VICAR User's Guide, Version 2, JPL Internal Document D-4186, Jet Propulsion Lab., California Institute of Tech., Pasadena, CA, 1989.

J. S. Pemberthy and D. S. Weld, "UCPOP: A Sound Complete, Partial Order Planner for ADL," Proceedings of the 3rd International Conference on Knowledge Representation and Reasoning, Oct 1992.

M. Stefik, "Planning with Constraints (MOLGEN: Part 1)," Artificial Intelligence 16,2(1981), pp. 111-140.

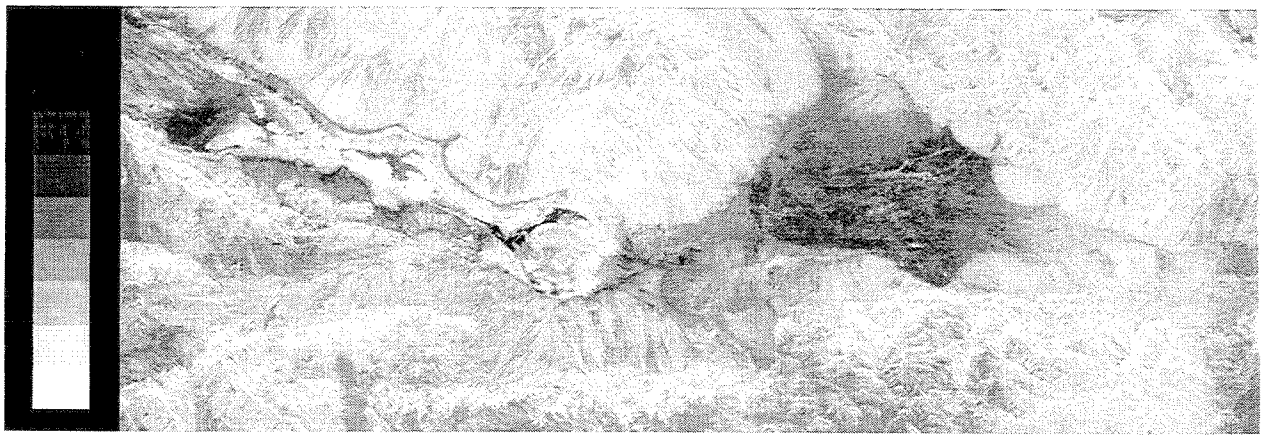


Figure 8: Aerodynamic Roughness Length Map Produced Using ASIP

Resource Re-scheduling for a Network of Communications Antennas

S. Chien, R. Lam, and Q. Vu
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive, M/S 525-3660, Pasadena, CA 91109-8099
{steve.chien, raymond.lam, quoc.vu}@jpl.nasa.gov

Abstract

This paper describes the Demand Access Network Scheduler (DANS) system for automatically scheduling and rescheduling resources for a network of communications antennas. DANS accepts a baseline schedule and supports rescheduling of antenna and subsystem resources to satisfy tracking goals in the event of: changing track requests, equipment outages, and inclement weather.

1. Introduction.

The Deep Space Network (DSN) [2] was established in 1958 and since then it has evolved into the largest and most sensitive scientific telecommunications and radio navigation network in the world. The purpose of the DSN is to support unpiloted interplanetary spacecraft missions and support radio and radar astronomy observations in the exploration of the solar system and the universe. There are three deep space communications complexes, located in Canberra, Australia, Madrid, Spain, and Goldstone, California. Each DSN complex operates four deep space stations -- one 70-meter antenna, two 34-meter antennas, and one 26-meter antenna. The functions of the DSN are to receive telemetry signals from spacecraft, transmit commands that control the spacecraft operating modes, generate the radio navigation data used to locate and guide the spacecraft to its destination, and acquire flight radio science, radio and radar astronomy, very long baseline interferometry, and geodynamics measurements.

From its inception the DSN has been driven by the need to create increasingly more sensitive telecommunications devices and better techniques for navigation. The operation of the DSN communications complexes require a high level of manual interaction with the devices in the communications link with the spacecraft. In more recent times NASA has added some new drivers to the development of the DSN: (1) reduce the cost of operating the DSN, (2) improve the operability, reliability, and maintainability of the DSN, and (3) prepare for a new era of space exploration with the New Millennium program: support small, intelligent spacecraft requiring very few mission operations personnel[4].

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

This paper appears in the working notes of the 1997 NASA Workshop on Planning and Scheduling for Space.

This paper describes the DANS system for rescheduling and resource allocation for antenna and subsystem resources in the DSN. DANS works from an initial schedule and uses prioritized pre-emption and localized search to find antenna and other equipment resources required to support changes to schedule requirements which may be caused by a wide range of circumstances including: changing track requirements from the flight projects, equipment outages, and inclement weather.

This paper is organized in the following manner. We begin by characterizing the current mode of operations of the DSN. Then we begin the main body of the paper which describes the DANS scheduling system. We begin by describing the problem representation within the DANS system. We then describe the priority-based pre-emption scheduling algorithm. Next we provide an example of the algorithm performing rescheduling. Finally, we describe the various reasons why rescheduling occurs.

2. Overview

Each week, a complex matching process between spacecraft project communication service requests and NASA Deep Space Network (DSN) resources occurs. In this process, project requests and priorities are matched up with available resources in order to meet communications needs for earth-orbiting and deep space spacecraft. This scheduling process involves considerations of thousands of possible tracks, tens of projects, tens of antenna resources and considerations of hundreds of subsystem configurations. Once this initial schedule is produced (8 or more weeks before implementations), it undergoes continual modification due to changing project needs, equipment availability, and weather considerations. Responding to changing context and minimizing disruption while rescheduling is a key issue.

The high level resource allocation problem for the smaller DSN antennas (26M and smaller) is currently handled by the OMP[6] scheduler. OMP accepts generalized service requests from spacecraft projects of the form "we need three 4-hour tracks per week" and resolves conflicts using a priority request scheme to attempt to maximize satisfaction of high priority projects. OMP deals with schedules involving thousands of possible tracks and a final schedule involving hundreds of tracks.

More recently, there has been a desire to automate the scheduling of the larger (34 and 70 m.) antennas. This paper describes the Demand-based Automated Network Scheduling System (DANS), an automated scheduling system being developed at the Jet Propulsion Laboratory (JPL) to schedule DSN 34 and 70 meter antenna resources. DANS is an evolution of the highly successful OMP system.

DANS uses priority-driven, best-first, constraint -based search and iterative optimization techniques to perform priority-based rescheduling in response to changing network demand. DANS first considers the antenna allocation process, as antennas are the central focus of resource contention. After establishing a range of antenna options, DANS then considers allocation of the 5-13 subsystems per track out of the tens of shared subsystems at each antenna complex (signal processing Center (SPC)) used by each track. DANS uses constraint-driven, branch and bound, best first search to efficiently consider the many possible subsystems schedules. Like the GPSS [1, 8] and SPIKE [5] systems which schedule space shuttle ground processing and science observations for the Hubble Telescope, DANS uses iterative repair and local search to repair (improve) schedules. However DANS also uses temporal constraint network and SIMPLEX techniques to efficiently reason about metric time constraints and to optimize time sensitive utility functions. Other work on scheduling ground stations by GSFC [10,11] has focused more on schedule editing rather than automate scheduling.

The DSN domain contains many resources. In the existing configuration (as of July 1996), it consists of 11 SPCs, 45 antennas, and 161 subsystems located at different sites around the world. The majority of the antennas can be classified as 26, 34, and 70 meter antennas. The 26 meter antennas on average handles 600 activities per week. The 34 and 70 meter antennas performs over 200 activities per week. Additionally, this workload is expected to increased dramatically in the next several years.

3. Domain Characteristics

In addition to the basic antenna resource allocation problem, the DSN scheduling problem is further complicated by three factors: (1) context-dependent priority; (2) subsystem allocation; and (3) the possibility of reducing the length of the tracks. DSN track priorities are context dependent in that they are often contingent on the amount of tracking the project has received so far in the week. For example, a project might have priority 3 to get 5 tracks, priority 4 to get 7 tracks and priority 6 to get 9 tracks (where lower priorities represent more important tracks). This reflects that 5 tracks are necessary to maintain spacecraft health and get critical science data to ground stations; 7 tracks will allow a nominal amount of science data to be downlinked; and 9 tracks will allow for downlinking of all science data (e.g., beyond this level additional tracks have little utility). An important point is that specific tracks cannot be simply labeled with these priorities (e.g., the project is allowed to submit 5 tracks at priority 3, 2 at priority 4 and so on). Rather when considering adding, deleting, or moving tracks the scheduler must consider the overall priority of the project in the current allocation context.

In addition to allocating antennas, DSN scheduling involves allocating antenna subsystems which are shared by each Signal Processing Center (such as telemetry processors, transmitters and exciters). Allocating these complicates the scheduling problem because it adds to the number of resources being scheduled and certain subsystems may only be required for parts of the track.

Finally, the DSN scheduling problem is complicated by the fact that the track duration can be relaxed. For example, a project may request a 3 hour track but specify a minimum track time of 2 hours. When evaluating potential resource conflicts the scheduler must consider the option of shortening tracks to remove resource conflicts. Currently OMP and DANS use a linear weighting scheme in conjunction with a modified SIMPLEX algorithm to trim tracks in accordance with prioritizations.

DANS accepts two types of inputs: 1. an 8-week prior-to-operation schedule from the Resource Allocation and Planning (RAP) team[1], and 2. activity requests from each individual flight project. The 8-week schedule is only a baseline for creating a conflict-free schedule. Many scheduled activities at that time are tentative at best, and subject to revision due to changing project status. Also, the schedule is for the antenna resources only; DSN subsystem scheduling is not considered at all in the 8-week schedule.

The activity requests are used by the flight projects to add and delete activities on an existing schedule due to changing project requirements and/or resource availability. The DANS objective is to satisfy as many activity requests as possible while maintaining a conflict-free status with minimum disruption to the existing schedule. DANS is intended for use by the operation personnel to maintain and update the DSN schedule throughout each schedule.

Another issue is the placement of activities onto the schedule. The possible times for a spacecraft track are limited by spacecraft orbit views, which are the periods in which the spacecraft is visible from a ground station. Also, the range from the antenna to spacecraft dictates the quantity and types of antenna(s) required for each activity. Sometimes, an array of multiple antennas instead of a single one is required to communicate with the spacecraft. In addition, the uplink and downlink activities can occur on different antennas, and can be several hours apart imposing additional dependencies between activities.

There are two types of activities in the DSN domain: spacecraft activities and ground activities. Spacecraft activities are submitted by projects and used to interact with spacecraft. They are required to satisfy the domain constraints above. Ground activities represent hardware maintenance. Antenna time which is not occupied by spacecraft activities is used for ground activities such as non-regular maintenance requirements and testing, with maintenance having higher priority.

Each DSN spacecraft activity is divided into 3 steps: pre-calibration (precal), tracking, and post-calibration (postcal). The time periods for each step are specified in ranges of

[1] The RAP team uses its own forecasting and scheduling systems [3,7] to establish and revise this baseline schedule.

values. The time periods are unique for each activity type, and depend on the antenna type and subsystem usage. DANS models this dependency by either shrinking or shifting activities to maximize resource utilization as dictated by activity type, antenna type, and subsystem usage.

DANS is required to schedule two different kinds of DSN resources: antennas and subsystems. Antennas and subsystems are unit resources and as such can not be shared by more than one activity. Subsystem resources are hardware such as transmitters which are required to work with an antenna during communication. Normally, there are many pieces of hardware that support each antenna and DANS is required to generate a schedule which allocates all necessary subsystems.

4. Resource Representation

The DSN consists of many Signal Processing Centers (SPC) situated around the globe. Each SPC may contain one or more antennas and many associated subsystems. These subsystems are used in conjunction with the antennas to perform tracks. DANS represents these resources in a hierarchical manner. Each SPC contains one or more antennas, which are the children of the SPC. There are also many SPC subsystem resources residing also as children of the SPC. For each antenna in the hierarchy tree, there are many DSS subsystems associated with it.

All the DSN resources are represented as capacity timelines which model a resource's usage at any instant of time for the duration of a schedule. A timeline is a sequence of capacity units each of which represents a constant resource usage within a time period. Shown in Figure 3 is the timeline representation of the DSS-26 antenna resource.

CapacityTimelines are constantly being modified and updated during the inference process to reflect the state of the resource at that instant of time. For performance reasons, a time slice caching scheme is used to expedite the query process. The scheme divides the timeline into a number of buckets. For timeline operations, the system finds the bucket that contains the moment first, and then changes the appropriate value. For example, when the system looks for a CptyUnit which contains the 1:30 moment as shown in Figure 3, it first identifies Bucket #2 as the container which includes the 1:30 moment. Then it traverses down the timeline starting from the beginning of Bucket #2 until it locates the CptyUnit that contains the 1:30 moment.

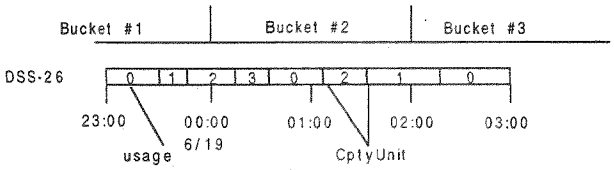


Figure 3: Capacity Timeline Resource Representation

5. Inference Engine

The current version of DANS employs a priority-based inference strategy to satisfy the project requests. The major bottleneck of the DSN domain is the antenna resources. This is due to the fact that there are limited numbers of antennas available while there are many identical subsystem resources that support unexpected events. In order to leverage this domain characteristic, DANS first focuses on antenna scheduling then resolves subsystem constraints.

The DANS scheduling process involves hypotheses generation, conflict identification, and conflict resolution. The process of scheduling a single activity consists of three major steps. First, the system generates an exhaustive list of solutions for each activity request at the antenna level. Second, this solutions list is then applied to the subsystem level to pick the best solution. Finally, the activity is placed onto the schedule. If the activity addition causes another activity deletion, the deleted activity(ies) will be rescheduled immediately. For ground subsystem maintenance activities which do not require antenna resources, the scheduling process will skip the first step(antenna scheduling) and start from the second step(subsystem scheduling). The antenna and subsystem inferencing flowcharts are shown in Figures 4 and 5.

At the antenna inferencing level, the system first selects an activity request (ACT) and extracts from the request the temporal window for the request (the time during which the project has requested a track). This window is then matched up with overlapping orbit views for the spacecraft (an orbit view is a time period during which a specific antenna can physically view the spacecraft), producing a list of valid intervals within the window. For each possible interval, the system tries to schedule the ACT. When a conflict, the system first tries to shift the ACT within the interval. If this action does not resolve the conflict and the conflicting activities have lower priority, the system identifies the conflicted activity(ies) for deletion.

While DANS is searching for possible ways to satisfy a request, it tracks the cost of each solution. The solution costs reflects the number of tracks which the new track displaces. Because scheduling further details of a track (more of the required subsystems) can only introduce more conflicts as a track becomes more completely scheduled its cost can stay the same or increase but never decrease.

DANS seeks to minimize the displaced tracks because the displaced tracks at best will be moved and at worst will not be accommodated in the final schedule. Either of these outcomes is bad, it is extremely disruptive to the projects adjust their operations in response to a moved track. Indeed, it is even worse if a track that a project was counting on is suddenly deleted. Thus, minimizing movements and deletions of tracks is key.

At the DSN subsystem inferencing level, DANS first identifies all the subsystems which are required to support the ACT. Then DANS selects the first available antenna

solution and tries to schedule the ACT to each of the subsystems at the specified time slot.

If conflict exists, it will try to resolve it as described above. When a solution exists, the system calculates the completed solution cost for both the antenna(s) and subsystems, and compares them to the previous completed solution costs and next antenna solution. If the current solution cost is less than or equal to the other solution costs, the system will commit to this solution, and will schedule the ACT to this specified time slot. Otherwise, this solution will be saved for future reference.

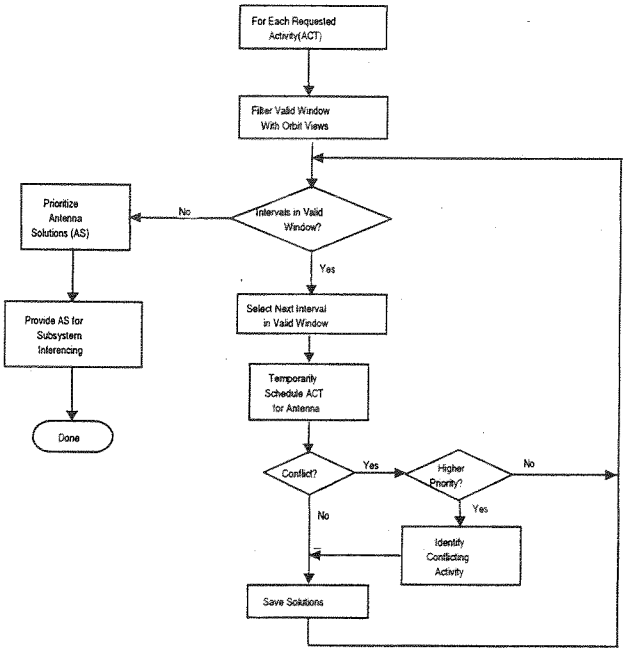


Figure 4: Priority-Based Antenna Inferencing Flowchart

After the system evaluates all the antenna solutions at the subsystem level, it will pick the best solution (i.e. the lowest cost solution to schedule the activity request). If this action requires deletion of other lower prioritized activities, these deleted activities will be submitted back to the schedule as a request immediately. DANS uses an equation to calculate each solution's cost. The equation is as follows:

$$\text{Solution Cost} = \frac{\text{NAD} * \text{activity priority}}{\text{NAD} - (\text{NAD} - 1) * 0.1}$$

where NAD = number of deletions required to schedule the current activity.

The cost is based on the activity priority. When there is no deletion required for scheduling the ACT, the cost is zero. When one deletion is required, the cost is equal to the activity priority. When there is more than one deletion required in order to schedule the ACT, the cost increases with the number of deletions.

6. Priority-Based Rescheduling: An Example

Consider the following example of the DANS scheduling algorithm. Initially, the DSS-14 antenna and its subsystems have committed their resources to two activities between 6:00am and 10:15am. Activity P0 has a valid window from 7:45am to 12:45pm, and occupies the 7:45am to 8:45am time slot. Activity P1 has a valid window from 9:15am to 12:45pm, and occupies the 9:15am to 10:15am time slot. Both P0 and P1 are DSN ground activities with priorities equal to 4. Activity P6 is a Galileo activity which requests a two hour duration between 5:45am and 10:15am on the

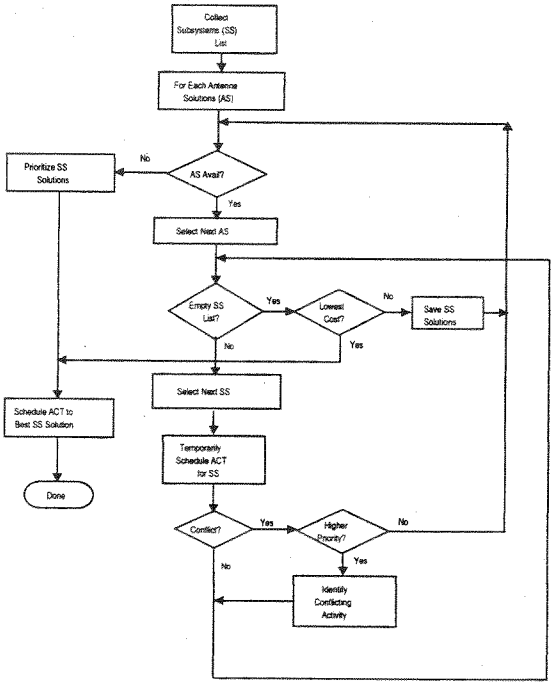


Figure 5: Priority-Based Subsystem Inferencing Flowchart

DSS-14 resources. P6 has a priority value of 3, which is higher than the priority of both P0 and P1. Subsequently, P6 can bump these two activities from the timelines when conflict arises. This information is shown in Table 1.

The DANS objective is to commit DSS-14 and subsystem resources to P6 activity and to maintain the conflict-free schedule with minimum disruption to the existing schedule. See Figure 6 for the scheduling sequences for this example.

For the Galileo P6 request, DANS first identifies all orbit views which are subsets of the P6 valid window. This enables DANS to filter out the invalid gaps and limits the search space. For this example, there is only one orbit view existing from 6:00am to 10:00am. Then the system turns its attention to the critical antenna resource to generate hypotheses. It traverses within the valid orbit view duration on the DSS-14 antenna timeline to identify time slots which can satisfy the 2 hour duration constraint. There are two valid time slots: Time Slot 1 from 6:00am to 8:45am; and

Time Slot 2 from 7:45am to 10:00am. DANS schedules P6 to both Time Slot 1 and Time Slot 2 to create two hypotheses. When P6 is place at Time Slot 1 for hypothesis 1 (HY1), it causes conflict with P0. Since P6 has higher priority than both P0 and P1, placement of P6 within this duration will delete activity P0 and the antenna solution cost becomes 4. When P6 is placed at Time Slot 2 for hypothesis 2 (HY2), it causes deletion of both P0 and P1, and the antenna solution cost is 4.21053. The system then sort all the hypotheses based on the antenna solution cost in ascending order. The result guides the inference process at the subsystem level without the necessity of performing an exhaustive search.

Activity	Project	Priority	Orbit View	Request Duration	Valid Window	Assignment
P0	DSN	4	N/A	60	7:45-12:45	7:45-8:45
P1	DSN	4	N/A	60	9:15-12:45	9:15-10:15
P6	GLLO	3	6:00-10:00	120	5:45-10:15	

Table 1: Example Activities Description

The system then continues the conflict identification at the subsystem level for both hypotheses. Activity P6 requires seven subsystem resources to accomplish the task. They are the LMC, SLE, TGC-A, MDA, NAR, RCV, and S-TWM. DANS identifies resource conflicts with P0 for the LMC, MDA, RCV subsystems for both hypotheses. The combined solution cost for the HY1 becomes 8.28571. This combined cost is then compared to the HY2 antenna cost, which is 4.21053. If the combined cost would have been less than the antenna cost value, the system would stop here and select the current hypothesis as the best solution. Since this is not the case, the system continues on the next hypothesis and calculates the combined cost for HY2 as 8.53485. Based on the result, DANS selects the first hypothesis as the solution since it has the lowest combined cost. It schedules P6 to the 6:00am to 8:00am duration and deletes P0 from the resource timelines. The system applies the same process to reschedule P0 activity. It identifies 3 time slots to generate hypotheses as shown in Figure 6. The system identifies the first time slot to schedule P0 between 8:00am and 9:15am with zero cost. It stops here having completed its task successfully to place the Galileo activity on the timeline without deleting any existing activities from the schedule.

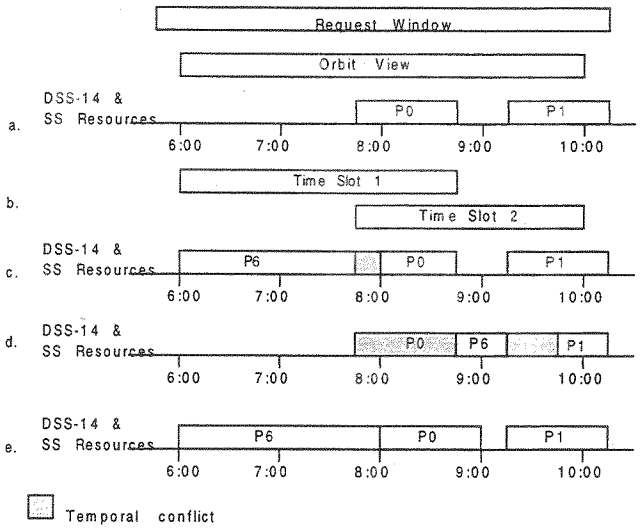


Figure 6: Priority-Based Scheduling Example; a) initial condition; b) valid time slots; c) hypothesis 1 causes P0 conflicts; d) hypothesis 2 causes P0 & P1 conflicts; e) final schedule after placing P6 at 6:00am and rescheduling P0 to 8:00am.

7. Rescheduling Context

Rescheduling DSN tracks is often necessary due to: equipment outages, last minute track requests, last minute changes to scheduled tracks, and atmospheric conditions impact on tracking capabilities. Rescheduling can occur in two ways: (1) it can be initiated top-down due to a change to a previously scheduled track or addition of another request; and (2) it can occur bottom-up in that equipment outages can occur or tracks can fail necessitating rescheduling. In the event of a new or modified request, DANS uses localized search to consider alternative methods for satisfying the new request (as previously described). This search uses as its bounding function a disruption cost measure which accounts for the overhead involved in moving already scheduled tracks and also a satisfaction measure accounting for what level of requests have been satisfied. Because we use branch and bound techniques DANS can guarantee that it will provide a reschedule optimal with respect to the combined disruption and satisfaction cost function.

In the event of a change in equipment availability, we are examining two solution methods. In both methods DANS first updates resource timelines to reflect the new resource level. Then, depending on the size of the change there are two options. First, if the change is localized DANS can perform branch and bound search (using the previously described cost function) to re-evaluate requests in light of the new equipment situation. However, if the change is too large in scope This exhaustive search is intractable. If an antenna goes down for a several day period the cascading effect on tracks can be quite great and thus rule out exhaustive search techniques. In these cases DANS can

instead first performs prioritized pre-emption to remove low-priority tracks to remove conflicts (by removing the lowest priority tracks participating in each conflict) and then re-evaluate project requests. This approach requires far less search but can produce suboptimal results (with respect to the twin goals of minimizing disruption and maximizing request satisfaction).

8. DANS LEO-T Demonstration

The DANS rescheduling system is currently being considered for two DSN scheduling applications: Network wide Network Preparation and Planning (NPP) datasets and also scheduling a simulated network of automated terminals designed to service low earth orbiting antennas (LEO-T). In this section we describe the demonstration of DANS on the simulated network of LEO-T antennas.

As part of an investigation into alternative low-cost means of providing communications services, JPL has been investigating the feasibility of fielding a network of small (3-meter) highly automated antenna stations which would use a resident workstation to operate the antenna and be capable of unattended normal operations [9]. This network would need to be scheduled automatically in order to fully automate communications services to low earth orbiting spacecraft. Such a scheduler would accept as inputs a set of requests for tracks from the projects supported by the LEO-T network and allocate coverage slots to be provided by the LEO-T stations.

In a demonstration of the DANS scheduling system we have scheduled a simulated LEO-T network, allocating tracks in response to real projects currently being supported by the regular DSN 26 Meter subnetwork. we simulated five LEO-T sites, using actual candidate sights in order to force realistic satellite to ground station geometries. These sites were: Pasadena, CA, Fairbanks, Alaska, Guam, US Protectorate, Spitzbergen Norway, and Kourou Africa. We then used five existing supported low-earth orbiting projects as the simulated project users of the LEO-T network: sampex-1, solar-a, strv-1a, strv-1b, and topex-1. Using existing orbiting projects allows us easy access orbital pattern and request data.

For each of these projects, we then generated requests that each project be covered to the maximal extent possible (i.e. each project requested continuous coverage from all possible stations). This was implemented by submitting a tracking request for each possible spacecraft to ground station view. In some cases where view periods were quite long (e.g., for the STRV-1a and STRV-1b projects) we artificially segmented the each view period into 3 equal time tracking requests in order to allow the LEO-T stations greater flexibility in covering spacecraft.

In the LEO-T demonstration, DANS used a very simple incremental schedule construction algorithm which is shown below.

```
for each project p in P
  for request r in p
    attempt to place r
    if r causes a conflict
    then remove the lowest priority
    track participating in the conflict
```

As the algorithm shows, DANS simply schedules the requests by considering each request and strongly preferring the highest priority requests. Note that this simple scheduling algorithm does not reflect that projects do not really need all of the possible tracks and that a projects priority is generally a function of the number of tracks that it has received so far. Using this simple algorithm, the DANS scheduler was able to solve the problems very quickly, the total problem includes consideration of hundreds tracking activities and 5 resources (the LEO-T stations themselves). DANS was able to generate the schedule in tens of CPU seconds running on a Sparcstation 20 with 64MB RAM.

9. Conclusions

This paper has described DANS, an automated scheduling system being developed to schedule DSN resources. DANS uses localized search and priority-based pre-emption to perform priority-based rescheduling in response to changing network demand. DANS first considers the antenna allocations, then considers allocation of the 5-13 subsystems per track (out of the tens of shared subsystems at each antenna complex) used by each track. DANS uses localized priority-driven, best first search to efficiently consider the large set of possible subsystems schedules.

References

[1] M. Deale, M. Yvanovich, D. Schnitzius, D. Kautz, M. Carpenter, M. Zweben, G. Davis, and B. Daun, "The Space Shuttle Ground Processing System," in *Intelligent Scheduling*, Morgan Kaufman, San Francisco, 1994.

[2] Deep Space Network, *Jet Propulsion Laboratory Publication 400-517*, April 1994.

[3] G. Fox and C. Borden, "Low Earth Orbiter Resource Allocation and Cpacity Planning for the Deep Space Network Using LEO4CAST," *Telecommunications and Data Acquisition* 42-118, pp. 169-178, August 1994.

[4] R. W. Hill, Jr., S. A. Chien, K. V. Fayyad, C. Smyth, T. Santos, and R. Bevan, "Sequence of Events Driven Automation of the Deep Space Network," *Telecommunications and Data Acquisition* 42-124, October-December 1995.

[5] M. Johnston & S. Minton, "Analyzing a Heuristic Strategy for Constraint Satisfaction and Scheduling," in *Intelligent Scheduling*, Morgan Kaufman, San Franc., 1994.

[6] E. Kan, J. Rosas, and Q. Vu, "Operations Mission Planner - 26M Users Guide Modified 1.0", *JPL Technical Document D-10092*, April 1996.

[7] S. J. Loyola, "PC4CAST - A Tool for Deep Space Netowrk Load Forecasting and Capacity Planning," *Telecommunications and Data Acquisition*, 42-114, pp. 170-194, August 1993.

[8] M. Zweben, B. Daun, E. Davis, & M. Deale, Scheduling and Rescheduling with Iterative Repair, in *Intelligent Scheduling*, Morgan Kaufman, San Fran., 1994.

[9] N. Golshan, W. Rafferty, C. Ruggier, M. Wilhelm, B. Haggerty, M. Stockett, J. Cucchisi, and D. McWatters, Low Earth Orbiter Demonstration Terminal, *Telecommunications and Data Acquisition 42-125*, 1996.

[10] Weinstein, S. C. Whitney, D. Zoch, M. Tong, "Developing a Scheduling System in Ada-Problems and Lessons Learned." *Fourth Annual NASA Ada User's Symposium Proceedings*. Hampton,VA: NASA/Langley Research Center, 1992.

[11] Wilkinson, G. R. Monetleone, S. Weinstein, M. Mohler, D. Zoch, and M. Tong, "Achieving Reutilization of Scheduling Software through Abstraction and Generalization." *Computing in Aerospace 10 Proceedings*. Washington, DC: American Inst. Aero. & Astro., 1995.

Maintenance Request Generation, Planning, and Scheduling for Highly Reusable Space Transportation

Steve Chien, Dennis DeCoste, Alex Fukunaga,
Gregg Rabideau, Tobias Mann, David Winkler
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, Pasadena, CA 91109-8099
firstname.lastname@jpl.nasa.gov

Abstract

This paper describes a preliminary concept demonstration of an automated capability to analyze telemetry data to generate maintenance requests and use this information to generate a maintenance plan. In this capability, real-time downlinked telemetry data from a reusable spacecraft would be analyzed to determine appropriate maintenance goals. These goals would then be used to develop a maintenance plan to be implemented upon arrival of the spacecraft. This capability would allow pre-positioning of resources and would lead to shorter turnaround times for the vehicle, increasing the fleet utilization rate and thusly leading to lower costs.

The Highly Reusable Space Transportation (HRST) program targets development of technologies leading to highly reusable space transportation systems which will provide extremely low cost access to space [HRST95, HRST96]. As part of this program, we have been developing and demonstrating advanced scheduling systems for the rapid generation and revision of plans for maintenance and refurbishment of highly reusable launch vehicles. In this application, real-time telemetry downlinked either during flight or immediately after flight would be analyzed to automatically generate a set of maintenance requests. These maintenance requests would then be transformed into a refurbishment plan by an automated planning and scheduling system which would account for available equipment and resources as well as the intricacies of the refurbishment procedures of the highly complex propulsion systems (this operations flow is shown below in Figure 1).

The end target for automating this process is to allow a turnaround of several days for the HRST spacecraft to support a flight frequency on the order of one flight per 1-2 weeks. As a comparison, the space shuttle refurbishment process currently takes approximately 65 days with a flight frequency of once per 4 months. It is worth noting that while automated planning and scheduling is already used extensively in ground processing for the Space Shuttle [Zweben 1994, Deale 1994], one difference is that highly reusable space transportation is being designed with the central goals of: reducing the turnaround time for the unit, simplifying determination of required maintenance activities, and simplifying the required maintenance activities themselves. If the maintenance schedule can be generated using in-flight telemetry then the refurbishment process can be speeded even further by allowing for downlinking of requests for pre-positioning of equipment and resources to minimize schedule delay.

Once the actual maintenance plan has been generated, the planning tool continues to be of use in two ways. First, in many cases there can be several mutually exclusive maintenance activities which can be performed next. Via lookahead and critical path analysis, automated scheduling software can determine the next activities to enable the minimal makespan (overall schedule execution time). Second, as unexpected events arise (such as equipment failures, resource unavailabilities, and schedule slippage), the automated scheduling software has the ability to revise the schedule so as to minimize schedule disruption (movement of activities and resources from their original assignments) and schedule slippage (delay of the completion of the overall refurbishment).

In order to test and validate this technology

[0]This paper describes research conducted by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.
Appears in the Working Notes of the 1997 Workshop on Planning and Scheduling for Space Exploration and Science.

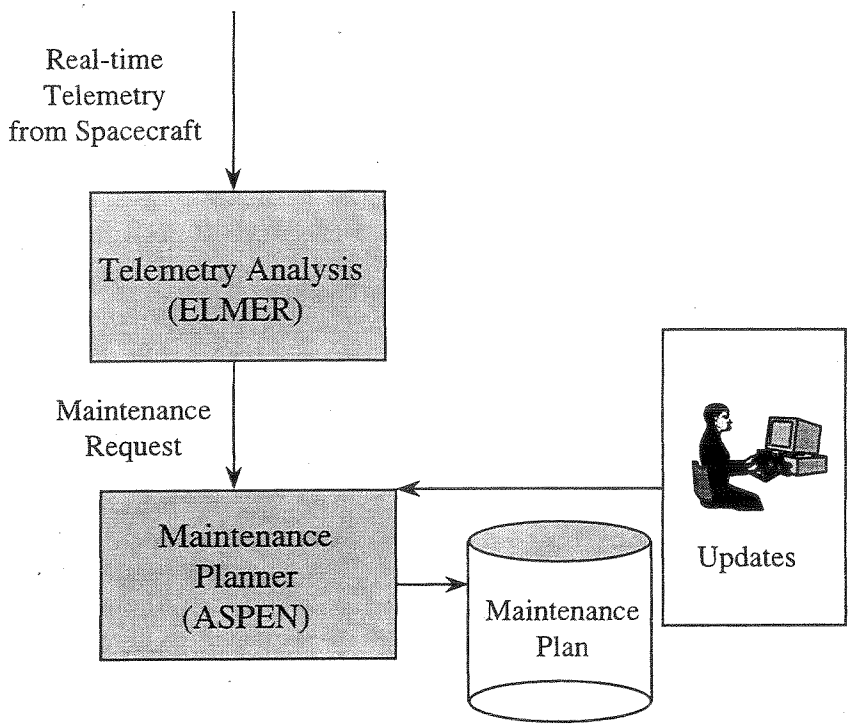


Figure 1: Information Flow for Automated Maintenance Request Generation and Planning

we built a preliminary proof of concept prototype which was demonstrated in the fall of 1996. This prototype consisted of two main modules (as previously described in Figure 1):

1. The telemetry analysis module, which analyzes real-time telemetry and produces a maintenance request; and

2. the maintenance plan generation module, which accepts the maintenance request and generates an appropriate maintenance plan.

For generating maintenance requests, we used the *Envelope Learning and Monitoring using Error Relaxation* (ELMER) system [DeCoste 1997a, DeCoste 1997b]. ELMER uses statistical machine learning techniques to learn and refine input-conditional limit functions from historic and/or simulated data. These limit functions define context-sensitive upper and lower boundaries, within which future data is expected to fall.

To test our prototype, ELMER was feed approximately 100 minutes of data for 12 sensors of the attitude control subsystem from a Space Shuttle mission. Unfortunately, the data we had did not contain abnormalities that were specifically relevant to the HRST domain. Thus, to functionally demonstrate our prototype, we defined (artificial) mappings from the abnormalities detected by ELMER in the Shuttle data to maintenance requests relevant to our HRST planner models.

In the development of the maintenance plan generation module for the prototype, we used the ASPEN planning and scheduling system [Fukunaga et al. 1997]. The ASPEN planning and scheduling system is a flexible, domain independent planning and scheduling application framework designed to support a wide range of planning and scheduling applications. For the HRST prototype, we acquired data on maintenance procedures for the LO2 and LH2 propulsion systems for the proposed Rockwell International led prototype for the X-33 Reusable Launch Vehicle program. While these procedures are not likely to be the same procedures used for HRST (since HRST is one generation beyond RLV) they are likely to have the same sorts of interactions, resources, and characteristics as other space transportation systems maintenance scheduling data.

Maintenance Request Generation Module

In our HRST prototype, ELMER is used essentially to classify each datum in the incoming engineering time-series data stream (i.e. for each sample time and each sensor) as either abnormal (i.e. outside the learned bounds) or normal. Each class of abnormality (e.g. a particular sensor being abnormally high) is then automatically mapped to a corresponding maintenance request, based on a simple manually-specified lookup table.

Traditional approaches such as expert systems could also provide such mappings. However, the key advantage of our approach is that ELMER allows operational definitions of our predeclared classes of symptoms to be automatically learned, based on historic data, to reflect context-sensitivity. Learning symptom detectors instead of manually specifying them can potentially be both cheaper and more accurate.

Given the large volumes of testbed and inflight data that are becoming increasingly common in many spacecraft domains, there is much promise that such learning can be feasible. The primary issue is a fundamental need to appropriately balance the cost of analyzing false alarms against the risks of missing significant faults.

Below we summarize some of the key ideas behind the ELMER techniques, with some emphasis on how it tries to avoid false alarms while still retaining sufficient accuracy.

Motivations Behind ELMER

In practice, automated monitoring of spacecraft relies heavily on limit-sensing and simulation. *Limit-sensing* compares time-evolving sensor values against high and low alarm limits. For the sake of minimizing the run time cost of monitoring, the numbers of false "nuisance alarms", and the costs of (manual) limit-determination, these limits are typically predefined, static, and relatively wide (e.g. "red-lines"). Alternatively, *simulation* dynamically computes tight limits — at the cost of requiring dynamic models that are expensive to create, test, maintain, and execute. Unfortunately, limit-sensing tends to be too imprecise (missing alarms) and simulation tends to be *too precise* (false alarms).

To help overcome those disadvantages, we developed ELMER. ELMER incrementally generates successively tighter high/low limit functions (*envelopes*), essentially moving from the wide static limits typical of limit-sensing toward the precise predictions of simulations, while avoiding unacceptable false alarm rates.

ELMER is motivated by our intuitions that simple limit-sensing *can* perform well, *if* we find limits that are *context-sensitive functions* and we limit-sense across *multiple perspectives* of the raw data. The intuition behind using multiple perspectives (e.g. learning bounds on both a raw sensor and on its derivative) is that that can allow each bound can be relatively loose (to reduce false alarms) yet collectively they might still detect most underlying faults. Supporting that intuition is that fact that many faults manifest themselves as many anomalous data across many times. Many tasks do not necessarily require detecting a fault as earlier as possible, especially if doing so risks costly false alarms.

In our early exposure to Space Shuttle operations, for example, we discovered that operators often slightly nudged the red-lines for some sensors up or down between flights, to account for both nuisance false alarms and for empirical realizations that some limits were excessively wide. ELMER can be viewed in part as a means for automating such a process via machine learning techniques.

Summary of ELMER Techniques

We define the *bounds estimation problem* as follows:

Definition 1 (Bounds estimation) *Given a set of patterns $\mathcal{P}$, each specifying values for inputs $x_1, ..., x_d$ and target y generated from the true underlying function $y = f(x_1, ..., x_D) + \varepsilon$, learn high and low approximate bounds $y_L = f_L(x_1, ..., x_l)$ and $y_H = f_H(x_1, ..., x_h)$, such that $y_L \leq y \leq y_H$ generally holds for each pattern, according to given cost functions.*

We allow any $1 \leq l \leq d$, $1 \leq h \leq d$, $d \geq 1$, $D \geq 0$, making explicit both our expectation that some critical inputs of the generator may be completely missing from our patterns and our expectation that some pattern inputs may be irrelevant or useful in determining only one of the bounds.

For each envelope sensor S, we compare its actual sensed value (y) at each time t+1 to its envelope predictions (y_H and y_L). To simplify discussion, we will usually discuss learning only high bounds y_H; the low bounds case is essentially symmetric. An *alarm* occurs when output y_H is below the target y, and a *non-alarm* occurs when $y_H \geq y$. We will call these alarm and non-alarm patterns, denoted by sets $\mathcal{P}_a$ and $\mathcal{P}_n$ respectively, where $\mathcal{N} = |\mathcal{P}| = |\mathcal{P}_a| + |\mathcal{P}_n|$.

ELMER relaxes the standard least squared regression cost function to include parameters which allow balancing the cost of (false) alarms against the cost of imprecision, as defined in Figure 2.

We can favor non-alarms (i.e. looseness) over alarms (incorrectness) by using values for the penalty (P) factors where $P_{H_a} > P_{H_n}$. This is analogous to the more common use of nonstandard loss functions to perform risk minimization in classification tasks.

$P_{H_a} = P_{H_n} = 1$ and $R = R_{H_a} = R_{H_n}$ gives us the standard class of Minkowski-R errors, where R=2 gives standard sum-of-squares error and R=1 gives classic *robust estimation* (which reduces the effects of outliers).

We focus here on linear regression (i.e. $y_H = \sum_{i=1}^{h} w_i x_i$) to perform bounds estimation, both for simplicity and because our larger work stresses heuristics for identifying promising explicit nonlinear input features, such as product terms. Nevertheless, this approach generalizes to nonlinear regression as well.

Since there is no general closed-form solution using such asymmetric cost, we use an iterative Newton method to perform batch linear regression. Conveniently, performance of the Newton method becomes analogous to that of the classic closed-form solution of singular-value decomposition (SVD) in the special case of symmetric cost (i.e. P_{H_n}=1, P_{H_a}=1, R_{H_n}=2, R_{H_a}=2 and P_{L_n}=1, P_{L_a}=1, R_{L_n}=2, R_{L_a}=2.). Even for asymmetric R and P values, we have found the weights resulting from SVD to be useful as initial seed weights. For example, such seeding allows P_{H_n}=0 to give a non-trivial result for lower-bounding. Otherwise, with initial weights of all 0, P_{H_n}=0 would result in zero error (since $E_{|L|}$ would be 0) and thus training would immediately stop with the constant result of y_L=0.

The complexity of linear regression is cubic in the number of inputs. For our application domains the number of sensors and transforms we wish to consider as inputs is typically in the hundreds or thousands. Thus, in practice ELMER must typically use heuristics to focus on promising candidate inputs. Indeed, given that the main goal in ELMER is not to necessarily fit the data precisely, suboptimal but cheaper and reasonable input subsets can often suffice.

We have found some simple greedy heuristics similar to classic forward subset selection techniques to be useful in practice. Specifically, we compute scores for each candidate sensor based on their correlations with the error residual of the current regression network. We add the candidate with highest correlation to the active input set and then rerun the batch Newton method to learn new (tighter) envelopes. We stop adding new inputs when the error on the validation data set starts to increase, since that suggests that overfitting is beginning to occur.

More fundamentally, our approach requires search over the space of possible and R and P values. Currently we have no optimal method of performing this search. Instead, we spot check combinations of various commonly useful values and pick the best result. For example, these choices typically include $R_{H_n} \in \{1, 2\}$, $R_{H_a} \in \{2, 10, 20\}$, $P_{H_n} \in \{1, 0.1, 0.01, 1e{-}3, 1e{-}4, 1e{-}5, 1e{-}10, 1e{-}15, 0\}$, and $P_{H_a} \in \{1, 1000\}$. The selection criteria involves not only the relative alarm versus non-alarm errors, but also other user preferences, such as acceptable levels of false alarm rates for each sensor. Once again, this does not result in optimal envelopes, but it does provide a cheap way to find reasonable ones. Furthermore, we have also been developing promising alternative techniques which avoid this search over R and P space altogether.

The Maintenance Scheduler

The maintenance planner portion of our prototype used the ASPEN planning and scheduling system. We now briefly describe the ASPEN planning and scheduling framework, the ASPEN modeling language, and the specific RLV model encoded.

The ASPEN System

ASPEN (Automated Scheduling and Planning ENvironment) is a reusable, configurable, generic planning/scheduling application framework designed to enable rapid development of automated scheduling systems for NASA applications,. An application framework [Pree 1995] is a class library (i.e., a reusable set of software components) that provides the functionality of the components found in prototypical instances of a particular application domain. Frameworks anticipate much of an application's design, which is reused in all applications based on the framework. In order to facilitate code reuse, ASPEN (as with many frameworks) incorporates multiple "design patterns". This implies a significant reduction in the amount of code necessary to implement successive systems.

The reusable components provided by ASPEN include:

- An expressive constraint modeling language to allow the user to naturally define the application domain;
- A constraint management system for representing and maintaining operability and resource constraints, as well as activity requirements;

$$e_H = \begin{cases} P_{H_n}(y_H - y)^{R_{H_n}} \text{ if } y_H \geq y \\ P_{H_a}(y_H - y)^{R_{H_a}} \text{ if } y_H < y \end{cases}, \qquad e_L = \begin{cases} P_{L_n}(y_L - y)^{R_{L_n}} \text{ if } y_L \leq y \\ P_{L_a}(y_L - y)^{R_{L_a}} \text{ if } y_L > y \end{cases}$$

$$E_H = \frac{1}{N}\sum_{p\in\mathcal{P}} e_H,\ E_{|H|} = \frac{1}{N}\sum_{p\in\mathcal{P}} |e_H|, \quad E_L = \frac{1}{N}\sum_{p\in\mathcal{P}} e_L,\ E_{|L|} = \frac{1}{N}\sum_{p\in\mathcal{P}} |e_L|$$

Figure 2: Asymmetric cost functions for high and low bounds.

Parameters: $P_{H_a}, P_{H_n}, P_{L_a}, P_{L_n} \geq 0$; $R_{H_a}, R_{H_n}, R_{L_a}, R_{L_n} \geq 1$.

- A temporal reasoning system for expressing and maintaining temporal constraints;
- A set of search engines for constructing, repairing, and optimizing plans/schedules;
- Linear Programming utilities for optimizing schedule preferences; and
- A graphical interface for visualizing plans and schedules (for use in mixed-initiative systems in which the problem solving process is interactive).

ASPEN is currently being utilized in the development of an automated planner/scheduler for commanding the New Millennium EO-1 satellite and a naval communications satellite, as well as for prototype schedulers for the ground maintenance for the Reusable Launch Vehicle and a design analysis tool for the Pluto Express spacecraft.

Models in ASPEN

In order to use the ASPEN scheduling system, a modeling language is used to specify domain-specific constraints and activities. Figure 3 shows part of a domain model specified in the ASPEN modeling language.

Within the ASPEN modeling language, the principal elements are: activities, resources, and states. Activities are the basic items being scheduled. In the RLV application, activities are maintenance steps required to refurbish the hardware, or setup steps necessary to support such operations. For example, each of the subsystems has refurbishment activities for various subsystems. Resources are finite elements whose capacity is limited and hence their use must be coordinated. Within the RLV domain, resources might consist of particular pools of workers with specific maintenance skills, physical locations which allow access to a piece of hardware (for example only two people may fit within a bay which allows access to a particular subsystem), or power limitations (not all maintenance equipment could be powered up at once). States correspond to physical or operational states of a system relating

```
Activity prevalve_removal {
  duration = [15, 20]
  slot subsystem ss
  after prevalve_prep with (ss = subsystem)
  before prevalve_replace with (ss = subsystem)
  reservation hydraulic_lift use 1
  reservation prevalve must_be purged
  reservation prevalve_area must_be illuminated
}

Resource hydraulic_lift {
  type non-depletable
  capacity 1
}

State_Variable prevalve {
  states purged unpurged
  transitions purged to_and_back unpurged
}
```

Figure 3: Sample of ASPEN modeling language (part of the Reusable Launch Vehicle maintenance model). This describes an activity for removing the prevalve of an engine subsystem.

to feasibility of activities. For example, in the RLV application, a state variable could model whether a hatch is open, whether a piece of equipment is powered on, or whether a valve is open.

To ground these concepts, the figure above shows a maintenance activity from the RLV model representing a prevalve removal step. This step must be preceded by a prevalve preparation step and followed by a prevalve removal step. This step also requires a hydraulic lift (a resource requirement) as well as two states (that the area be illuminated and that the prevalve be purged).

The exact maintenance request is derived from the engineering analysis performed by ELMER. Thus, depending on the performance data of the RLV, different maintenance requests would be generated, and hence different planning and scheduling problems posed to ASPEN. ASPEN would receive a maintenance request from the ELMER module, and then construct a plan/schedule using the available

resources.

The RLV Maintenance Scheduling Domain

In the prototype demonstration of the maintenance planning element, we utilized test maintenance procedures developed by Rockwell International during the Phase 1 competition of the Reusable Launch Vehicle Program which ended in July 1996. Specifically, we used the maintenance procedures developed for the LO2 and LH2 propulsion systems for the X-33 Reusable Launch Vehicle. These procedures were then modeled in the ASPEN planning and scheduling system [Fukunaga et al, 1997]. The procedures derived for maintaining and refurbishing the test articles provided a rich testbed for Space Propulsion System Maintenance Scheduling. In all, the testbed involved refurbishment of 2 major systems with 16 subsystems, with a total of approximately 700 lowest level activities in a complete (all subsystems) refurbishment plan. The model that we developed for schedule generation involved: 576 activity types, 6 resources, and on average 6 state, resource, and precedence constraints per activity. In this application we allowed maintenance requests to request either refurbishment of specific subsystems or major systems. In order to schedule the maintenance requests the ASPEN system used a forward sweeping greedy dispatch algorithm which used knowledge of the required precedences of activities in the plan. The resultant scheduler was able to generate schedules for smaller refurbishment problems (8 subsystems, 358 activities) in approximately 8 minutes.

Discussion and Conclusions

The current prototype does not substantially validate the viability of automated maintenance request generation and maintenance planning. Rather it merely serves to illustrate the possibility of such a concept. In order to validate the concept, a more comprehensive demonstration would need to be performed with several key differences. First, the request generation should use actual relevant engineering data, i.e. the same data which currently would be manually analyzed to determine maintenace requirements. Second, realistic mappings from engineering data to maintenance requests should be used. Finally, a more detailed and realistic model of maintenance activites should be implemented. A demonstration in this context would provide significant evidence of the promise of such an approach.

Other work in the area of maintenance scheduling includes work on space shuttle ground processing [Deale et al. 1994, Zweben et al 1994] and testbed work on RLV maintenance scheduling [Allen 1996].

This paper has described a proof of concept demonstration of automated analysis of engineering data to generate maintenance requests for highly reusable space transportation and automated planning and scheduling of such requests using an automated planner. This demonstration highlights the possibility of such an approach to reduce turnaround times for space transportation systems - thus allowing a higher utilization rate, reduced fleet size, and hence reduced operations costs. While the initial prototype was successful, significant further work is needed to validate the viability of the concept.

Acknowledgements

The authors gratefully acknowledge the assistance of John Allen of NASA Ames for his assistance in obtaining HRST testbed data and information.

References

[Allen 1996] J. Allen, Personal Communication, NASA Ames Research Center, 1996.

[Deale et al, 1994] M. Deale, M. Yvanovich, D. Schnitzius, D. Kautz, M. Carpenter, M. Zweben, G. Davis, and B. Daun, *Space Shuttle Ground Processing Scheduling System,* in M. Zweben and M. Fox (eds.), Intelligent Scheduling, Morgan Kaufman, 1994.

[DeCoste, 1997a] D. DeCoste, Automated learning and monitoring of limit functions. Proceedings of the Fourth International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS-97), Japan, July 1997.

[DeCoste, 1997b] D. DeCoste, Mining multivariate time-series sensor data to discover behavior envelopes. Proceedings of the Third Conference on Knowledge Discovery and Data Mining (KDD-97), Newport Beach, CA, August 1997.

[Fukunaga et al, 1997] A. Fukunaga, G. Rabideau, S. Chien, and D. Yan, ASPEN: A Framework for Automated Planning and Scheduling of Spacecraft Control and Operations, Proceedings of the International Symposium on Artificial Intelligence, Robotics, and Automation in Space (I-SAIRAS97), Tokyo, Japan, 1997.

[HRST95] Proceedings of the 1995 Technical Interchange Meeting on Highly Reusable Space Transportation, Huntsville, AL, July 1995.

[HRST96] Proceedings of the 1996 Technical Interchange Meeting on Highly Reusable Space Transportation, Huntsville, AL, August 1996.

[Pree 1995] W. Pree, *Design Patterns for object oriented software development.* ACM Press, 1995.

[Zweben et al. 1994] M. Zweben, B. Daun, E. Davis, and M. Deale, *Scheduling and rescheduling with iterative repair,* in M. Zweben and M. Fox (eds.), Intelligent Scheduling, Morgan Kaufman, 1994.

Emerging Techniques in Evolutionary Scheduling

David Corne
Parallel, Emergent and Distributed Architectures Laboratory
Department of Computer Science
University of Reading, UK
Email: D.W.Corne@reading.ac.uk

Abstract

This article briefly presents and discusses a selection of emerging ideas for approaches to scheduling optimisation problems. Those discussed mainly arise from the 'evolutionary computing' research community. They are: Optima Linking, Optima Contouring, an Immune System Approach, and 'Mutagenetic Algorithms'. What each of these methods have in common is simply that they are new, novel (modulo, in some cases, clear conceptual links with other established methods), seem promising for general schedule optimisation, and are as yet only sparsely described in the research literature, if at all. Each of the selected new techniques is actually a 'general', or 'weak' algorithm, but happen to be in development by researchers active in the scheduling domain. It is rather unclear whether any of these will endure to earn a reputable place in the library of scheduling optimisation tools, but for the moment they each certainly seem worth further investigation. Some of them are currently under investigation as part of a project by the author dealing with the massive nuclear waste reprocessing and maintenance scheduling problems faced by British Nuclear Fuels Ltd.

Introduction

This article briefly presents and discusses a selection of emerging ideas for approaches to scheduling optimisation problems. The selection is quite eclectic, and represents the authors' personal 'top four' solutions to a multiobjective function taking into account: novelty, general interest, and promise.

Evolutionary scheduling, in general, is a successful and fast growing research area which has already led to several practical systems in place [1]. There are several 'standard' evolutionary scheduling techniques which are now quite well documented and established. The essence of such an 'evolutionary scheduling technique' is a particular method for representing candidate schedules as 'chromosomes' and a particular set of methods for operating on such chromosomes to produce new candidate schedules. One of the new techniques briefly discussed, 'Mutagenetic Algorithms', is an extension of these established evolutionary techniques incorporating the idea of 'mutagens': special agents, themselves subject to artificial evolution, which perform specific operations on candidate schedules.

Two other of the discussed methods, Optima Linking and Optima Contouring, arise from the evolutionary computing research community in a slightly different way. They arise from certain observations about the structure of schedule optimisation fitness landscapes, noticed in [2], for example, which in turn lead to promising ideas for organising the direction of local search.

Another of the methods, the Immune System approach, is inspired by recent work (for example [3]), showing how ideas from the immune system can be applied in computer science and optimisation.

Each of these methods is relatively new, and seem promising for general schedule optimisation, and hence worth bringing to the attention of scheduling practitioners. In fact, however, each of the selected new techniques is really a 'general', or 'weak' method, but it happens to be the case that each is in development by researchers active in the scheduling domain.

No claims are made here as to whether any of these techniques will endure to earn a reputable place in the library of scheduling optimisation tools, but for the moment they each certainly seem worth further investigation. Some of them are currently under investigation as part of a project by the author dealing with the massive nuclear waste reprocessing and maintenance scheduling problems faced by British Nuclear Fuels Ltd.

In the remainder of this article, each are described in turn.

Mutagenetic Algorithms

What practically all evolutionary computation or local search applications have in common is the notion of a fixed set of 'operators'. For example, imagine a simple scheduling application in which a candidate solution can be represented as a sequence of 9 tasks, such as:

3 2 5 1 6 7 9 8 4

In the application, such a list of tasks might, for example, be interpreted as the input to a greedy schedule builder. A typical operator on such a list is the 'adjacent swap' operator. This involves choosing a random pair of adjacent tasks, and swapping them, perhaps changing the above into:

3 2 5 6 1 7 9 8 4

Very many other such operators exist. There are inversion operators, for example, which choose a random contiguous chunk of a sequence and reverse the ordering of tasks in that chunk. There are 'shunt' operators, which shift an entire chunk of a sequence from one place to another, and so on. Similarly, there are a variety of recombination (mult-parent) operators, which have some way of producing a 'child' sequence by somehow combining aspects of two parent sequences.

The idea of a *Mutagenetic Algorithm* (MA) is to add further complexity and specificity to the mutation operators in an evolutionary approach, taking inspiration from [4]. In [4], the role of 'mutagens' in natural evolution is discussed. These are biological vectors which perform specific, specialised mutation operations on chromosomes. In nature, mutagens are themselves subject to natural selection, and [4] develops the thesis that, loosely speaking, mutagens and their targets co-evolve over time with the result that mutagens become better at making mutations which are useful to their targets. A straightforward implementation of this idea in our simple 'sequence' example would be to have a structured class of mutation operators which each specialise in a particular set of operations. These operators vie with each other, and based on feedback concerning the successes or failures of their use, certain operators become more favoured and hence used more often.

A particularly simple example of such a structured class of operators is as follows:

Class: (X X X)
Instances: (1 2 5) , (3 6 8), (1 3 9), ...

In this case, a class is simply a list of three numbers, intended to refer to particular *tasks* in the sequence. An instance of this class, such as "(1 2 5)", describes a mutation operator which takes those particular tasks, wherever they happen to appear in the candidate chromosome, and randomly reorders them.

In general, a structured mutation operator, or 'mutagen', involves a pattern and an action. The pattern describes which elements of the chromosome it operators on, and the action describes what it does. In the above illustration the 'action' is simply assumed to be random reorder. Further possibilities for patterns and actions can be designed at will, of course.

The point of all this is the underlying notion that certain specific types of operation may be particularly useful in an instance of a general scheduling problem. With a mutagenetic algorithm, the idea is to unleash a collection of different such operators, and co-evolve these in line with the population of schedules, in order to draw out and use these particularly useful operators. There is a fair degree of intuitive and other evidence for this view. For example, one of the more successful recent evolutionary algorithm approaches to job shop scheduling problems makes particular use of operators which reorder the tasks along the critical path [5], borrowing from work recommending similar operators within other search techniques [6,7]. This focusses mutation operations towards a particular set of tasks, which itself varies between schedules, and over time, during the course of an algorithm. Whereas the the relative conceptual simplicity of a 'pure' job shop scheduling problem, however, allows the notion of 'tasks on the critical path' to be pinned down as ideal candidates for mutation, more complex and messy real-world problems are also likely to involve particular sets of task more fruitful for mutation in certain ways than others, but far less easily spotted by analytic or argumentative means.

A mutagenetic algorithm attempts to directly find such operators. In its 'adaptive' form. the mutagenetic algorithm will involve a continually changing set of such operators, adapting themselves to the stage of search. However, there is reason to explore a simpler, 'fixed' form, in which a particular set of specialised operators is designed in advance, and then used in ensuing search as a fixed collection. Preliminary evidence (to be further described in a later version of this article) is promising for both methods.

Optima Linking

Optima Linking is a new idea which has been found to show excellent promise in application to scheduling problems. As such, it joins the armoury of methods being researched for scheduling, but Optima Linking arrives with the extra benefit of being useful for prob-

ing the structure of scheduling landscapes.

The idea of Optima Linking has been developed independently by Darrell Whitley and by Colin Reeves. It is first described in Rana and Whitley [8], which tests it along with other methods on a suite of standard function optimisation tasks. The essential idea is to generate a path between two previously found local optima; a good point on this path then becomes the starting point for a new local search. Its merit seems clear from the intuitively agreeable notion that the basins of attraction of undiscovered optima are often likely to lie 'between' other local optima. Similar ideas underlay recent work by Glover [9], called 'path-relinking', by Reeves (personal communication), and by Yamada and Nakano [5], achieving excellent results on scheduling benchmarks. In more recent work, Whitley has found Optima Linking to produce better than previous known best results in a benchmark project management scheduling problem (personal communication). These impressive results from early use of Optima Linking and similar ideas are very promising.

In addition to its raw potential as an optimisation technique, Optima Linking is also distinctly useful for exploring fitness landscape structure. To a degree, of course, other search techniques are viewable in this light. Repeated runs of simple hillclimbing, for example, can indicate the relative preponderance of different local optima, and the sizes of their respective basins of attraction. With more sophisticated search methods, however, ability to inform of landscape characteristics is typically compromised by the complexity of the technique itself. The performance of repeated runs of a genetic algorithm on a scheduling problem, for example, may tell us that it was more successful at navigating the 'bumps' than some other method, but more precise landscape information becomes lost, being 'smoothed over' in the dynamics of the algorithm. Optima Linking, in contrast, delivers strong performance while retaining understandable dynamics; its performance is intimately related with the shape of the fitness landscape in a way which can directly suggest pertinent details of landscape structure. Therefore, research on the development of optima linking based algorithms has a natural potential to be able to explain its results and explain the relative successive of the techniques developed in a useful way. Via 'mapping' landscapes with simple Optima Linking, such explanation can also feed back usefully into understanding the relative performance of other techniques on the same problems.

A 'canonical' Optima Linking operation works by iterating the following process: 1) generate a pair of optima using a local search method. 2) Generate a path of solutions between these optima 3) Continue local search starting from the best solution found along this path.

There is clearly massive scope for potentially powerful variations on this theme which retain the essential idea. One tunable aspect is the number of initial points to begin linking from, and regime used to determine the choices of optima to link (analogous to population and selection in a population-based evolutionary algorithm). Another important variable aspect is the method used to generate the linking path. Rana and Whitley's results on function optimisation [8], for example, build the link by at each step finding the 'best' next move along the path. In contrast, [10] finds a faster and simpler Optima Linking method successful in which the path is made simply from successive random steps in the right direction. Meanwhile, the Optima Linking-like notion underlying the recombination operator in [5] uses a local search between two parents, where the search is slightly biased towards the direction of the 'destination' parent.

Further variable aspects of the Optima Linking idea concern the neighbourhood move operator used to generate the linking path, which need not be the same as the operator used in the local search component, the local search method itself, which need not remain fixed, and parametric aspects of the local search method. For example, it is not necessary for the pair of points to be linked to be strict optima; in [10], Corne shows that points generated from rather modest local search effort (and certainly not locally optimal) still yield linking paths with significantly fruitful result. This in itself suggests intriguing aspects of the locations and accessibilities of the basins of attraction of good optima.

Showing particular promise for scheduling, Whitley has investigated the use of an Optima Linking algorithm on a publicly available Resource Constrained Project Scheduling benchmark problem. This problem involves scheduling 575 jobs with precedence constraints and other complex restrictions.

Using a permutation-based representation, an 'adjacent-task swap' local search operator, and linking optima by linking their inverse permutations[1], a straightforward Optima Linking algorithm found a new best solution for this problem, and in fact found this solution in 70% of attempts.

Optima Contouring

Optima contouring is a variant of Optima Linking, differing in a rather critical respect: there is no 'linking' between 'optima'! It recognises the intuition and ap-

[1]The inverse of a permutation indicates, for each element i in turn, how many larger elements precede it in the permutation. Eg: the inverse of {3 0 2 1} is { 1 2 1 0 }

peal behind the notion that paths between diverse optima may yield new basins of attraction, but contends that the crucial aspect of such a path is that it *emerges from a local optimum*, rather than *goes between local optima*. Such a comment must of course be taken in careful context. The 'No Free Lunch' theorems [11] do well to remind optimisation researchers that the valid goal of our efforts is to find ideal matches between algorithms and problem classes, rather than to only find 'ideal algorithms'. Hence, the idea of Optima Contouring is to make a particular assumption about scheduling landscapes, (viz: that they are generally as observed by Mattfeld [2]) and use this assumption in its design. Simply put: Optima Contouring is designed to exploit landscapes in which the distribution of local optima is like a 'massif central', becoming gradually more numerous and densely packed with increasing fitness.

The way it works, inspired of course by Optima Linking, is to first find a single local optimum by local search, and then describe one or more contours (loci of points) at concentric, fixed distances around this optimum. The points found around these contours are used to choose the starting point for the next local search. This process simply iterates with each newly found local optimum. The use of a number of contours at different distances from the optimal point (rather than, say, using one contour and simply restarting search from the best point on it), is meant to avoid being deceived by local 'massif central' like structure, and hence search more successfully for the global massif central cluster. This is achieved by a computationally quick cluster analysis of the contour points, dealing with both genotypic distance and fitness, which yields the presumed most promising point from which to restart the search. In particular, the 'best' point found on a contour may not be chosen, since it may be outweighed by a more dense cluster of quite good points on a different 'side' of the previous optimum.

Early results with this idea on simple job shop scheduling problems seem to show that it improves on Optima Linking, particularly in terms of speed.

Immune System Methods

So far, we have looked at some techniques designed specifically for schedule optimisation with respect to the standard ranges of criteria, such as makespan. A very thorny problem in real world scheduling, however, which academic research has not really focussed on very much, is the need for continual re-scheduling. This translates into a need for *robust* schedules: solutions to scheduling problems which can be quickly and easily altered to provide new, good solutions when circumstances change in (at least) typical ways.

Current research at Edinburgh [12] is looking into an immune-system based method for robust schedule generation. The context is a dynamic job-shop scheduling problem in which jobs arrive continually, and the environment is subject to many unpredictable changes.

The human immune system defends the body against toxins by producing *antibodies*, which recognise foreign molecules (*antigens*). There are unimaginably many potential antigens, but the immune system is able to defend us from a very great number of different antigens with very limited resources. Similarly, in a scheduling situation, ideal solutions should be able to 'cope' with the wide range of potential disruptions that may occur. If we consider these possible disruptions (eg: late arrival of a required resource, breakdown of a particular workstation, etc ...) as antigens, then the natural furtherance of the mapping is to consider a schedule to be an antibody. We therefore ideally require a collection of possible schedules which together cover a great range of possible disruptions to the existing statement of the problem. In practice, this set of schedules can be used as a 'library'; initially, one schedule is put into operation. When a disruption occurs, we search through the schedule library to find a different schedule which can handle the now altered problem. This mirrors the immune system's search through its gene library to find antibodies which can handle a particular new foreign substance. Notice, however, that we require not only a workable new schedule from the library, but one which is maximally similar to the one previously in operation. The altered schedule can then be implemented with minimised overall disruption.

An immune system approach to robust scheduling therefore has the task of generating such a library of schedules, given standard scheduling problem data together with information about the typical changes that may occur. Early work on this notion is very promising [12]. In particular, it has been found to be at least as good (in terms of quality of replacement schedules) and far more efficient, than an earlier more standard range of techniques for rescheduling [13].

Summary

A handful of new techniques have been described which seem to show promise in scheduling. Naturally, this promise arises mainly from the fact that they in some way exploit ideas about the structure of scheduling landscapes. This is particularly true for Optima Linking and Optima Contouring, but also the case for Mutagenetic Algorithms, where, in its adaptive form, the idea is for the details of the operators to evolve over time so as to better grasp the neighbourhood structure. In its 'design' form. the Mutagenetic Algorithm idea is

to directly tailor the neighbourhood sampling scheme to the problem instance in question. Meanwhile, in the Immune System case, the idea is not to exploit scheduling neighbourhood structure (although there are clear opportunities and good reasons for doing so in the general framework), but to attack a particularly important aspect of real-world scheduling problems.

These are all fairly 'late-breaking' ideas, which have proved promising in early work. The hope and purpose of this article is to encourage further use and exploration of these ideas. A later version of this article will incorporate more detailed discussion and results on each of these techniques, applied to a particularly nasty real world scheduling problem.

References

[1] Corne, Ross, Practical Issues and recent Advances in Job and Open-Shop Scheduling, in Dasgupta and Michalewicz (eds), Evolutionary Algorithms in Engineering Applications, Springer, 1997. 0-61749-3, pages 39–49.

[2] Mattfeld, Evolutionary Search and the Job Shop, University of Bremen PhD thesis, 1996.

[3] Hightower, Forrest, Perelson, The Evolution of Emergent Organization in Immune System Gene Libraries, in Proceedings of the Fifth International Conference on Genetic Algorithms, pages 344-350, 1993.

[4] Wills, The Wisdom of the Genes: New Pathways in Evolution, Oxford University Press, 1991.

[5] Yamada, Nakano, "Scheduling by genetic local search with multi-step crossover", in *Parallel Problem Solving from Nature - PPSN IV*, Voigt, Ebeling, Rechenberg, Schwefel (eds.), Springer LNCS 1141, 1996.

[6] Laarhoven, Aarts, Lenstra, Job Shop Scheduling by Simulated Annealing, Operations Research 40(1), pp. 113–125, 1992.

[7] Brucker, Jurisch, Sievers, A Branch and Bound Algorithm for the Job Shop Scheduling problem, Discrete Applied Mathematics 49, pp. 107–127, 1994.

[8] Rana, Whitley, "Bit Representation with a Twist", in *Proc. of the 7th Int'l Conf. on Genetic Algorithms*, Morgan Kaufmann, 1997.

[9] Glover, Laguna, "Tabu Search". Chapter 3 of "Modern Heuristic Methods", Reeves (ed.), Blackwell Scientific, 1993.

[10] Corne, Ogden, "Evolutionary and Other Approaches to the Preacher Planning Problem", submitted to *PATAT2, Proc. of the 2nd Int'l Conf. on the Practice and Theory of Automated Timetabling*, Springer, 1998 (to appear).

[11] Wolpert, MacReady, No Free Lunch Theorems for Optimization, IEEE Trans. on Evolutionary Computation 1(1), pp. 67–82, 1997.

[12] Hart, Ross, Nelson, Producing Robust Schedules via an Artificial Immune System, submitted to the 1998 IEEE Conference on Evolutionary Computation, Alaska, 1998.

[13]
Hsiao-Lan Fang, "Genetic Algorithms in Timetabling and Scheduling", PhD Thesis, Department of Artificial Intelligence, University of Edinburgh, 1994

Planning and Scheduling for Regional Validation Centers

Robert F. Cromp
Code 935
NASA/Goddard Space Flight Center
Greenbelt, MD 20771
cromp@sauquoit.gsfc.nasa.gov

John R. Bane
Global Science and Technology, Inc.
6411 Ivy Lane, Suite 300
Greenbelt, MD 20770
bane@gsti.com

Abstract

Early in 1997, Code 935 at NASA/Goddard Space Flight Center installed the prototype version of its Regional Validation Center (RVC) system at four university sites. The RVC system is a framework for acquiring, processing, indexing, storing and retrieving satellite image data. This paper describes the planning and scheduling components of the RVC, which are responsible for sequencing algorithms to produce products and for assigning computational resources to those sequences.

Introduction

Over the past decade, computer science researchers in the Applied Information Sciences Branch, Code 935, NASA/Goddard Space Flight Center have developed an end-to-end scientific spatial database management system called the Intelligent Information Fusion System (IIFS) (Cromp, Campbell, & Short 1993)(Short *et al.* 1995) with the express purpose of developing, incorporating and evaluating state-of-the-art techniques for handling EOS-era scientific data challenges. Since 1989, the IIFS has been based on an object-oriented database which is used to store metadata about large scale data holdings. The metadata itself is organized to enable fast, efficient access to the appropriate data sets. To handle image data, our research group has developed an innovative spatial data structure known as a sphere quadtree (SQT) that more naturally represents data acquired globally. Additionally, we have developed a number of fast techniques for automatically extracting information about image content, enabling users to query for pertinent data sets based on the features of scientific interest within the images themselves. The IIFS includes a planner/scheduler/dispatcher module (the PSD) that monitors and assigns system resources for processing the data flow, including all processing of higher level products, and extraction and assembling of metadata.

The IIFS is a domain-independent search engine for managing large volumes of data. We have recently embedded the IIFS in an end-to-end remote sensing system which acquires satellite data as the satellite passes from horizon to horizon, and gives users the capability to query for data products using a graphical interface. The resulting system is known as a Regional Validation Center (RVC).

Earlier this year, Code 935 at NASA/GSFC installed a prototype version of the RVC system at four university sites (University of Maryland/Baltimore County, Clemson University, University of Southwestern Louisiana, and University of Hawaii). Each site acts independently as its own RVC, and customizes the system to match their own remote sensing applications.

The RVC system is intended to start out as a small, low-cost way to capture, process, index, store, and retrieve satellite image data, with the potential to grow in capability as its data store and the needs of its users grow. The initial configuration at our prototype sites consists of a COTS satellite dish with an associated PC-class control computer, a medium-power Unix workstation to process the data and store the metadata, and a small robot tape drive to store the data and products. If an RVC needs more storage or processing power, it can simply add more Unix workstations as resources (see below).

Users at an RVC organize their system around a central theme, customizing the RVC software by developing and registering specific algorithms which serve to organize and enable retrieval of the RVC's data holdings. Given the finite availability of computing resources, and the complexity of product chaining permissible in an RVC, it is essential that the appropriate planning and scheduling occur to satisfy the users in a timely manner.

The planner/scheduler must be able to respond in a goal-directed mode to handle users' queries, and in a data-driven mode whenever new data is ingested from a satellite pass. We present our approach at performing these tasks in the remainder of the paper.

The RVC Framework

System Components Figure 1 presents the components of the RVC and the general information flow between them.

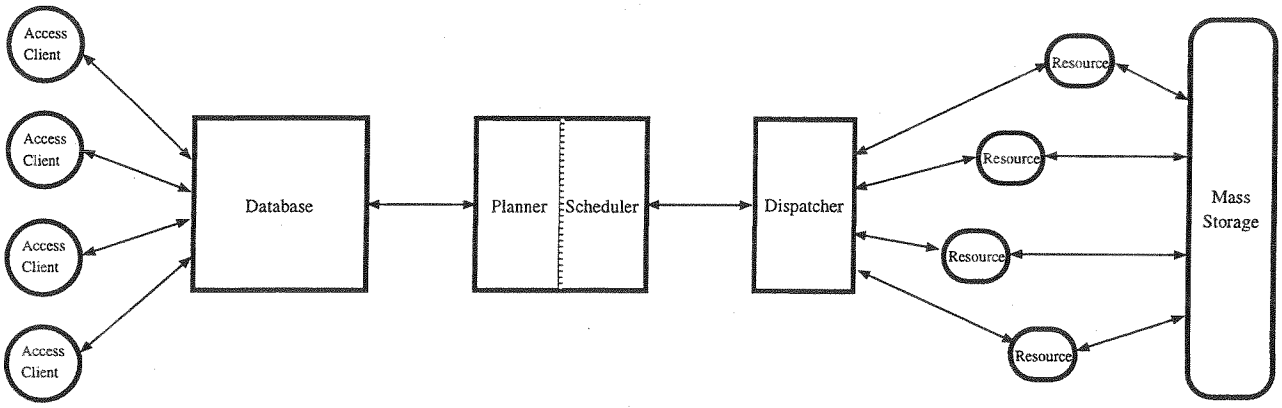


Figure 1: A simplified Block diagram of the components of the RVC system. Each block in the diagram represents a separate process; the RVC is a peer-to-peer system that coordinates the efforts of its parts by passing information via network sockets.

The components function as follows:

Access Clients - connect to the Database to submit queries and product requests. Currently we have implemented one access client for users called `rvci` and two maintenance/curator clients called `algo` and `resource`. The current access clients are written in Tcl/Tk.

Database - an object-oriented database holding the metadata for observations as well as descriptions of compute resources and programs the RVC can use to generate products. Actual image data is not stored in the database; it lives in the Mass Store and is referred to in the database via URL. The current RVC database is a C++ application using ObjectStore.

Planner - a system that accepts product requests and generates sequences of algorithms that will fulfill the requests. Product requests come in as lists of goals (types of products) and initial conditions (which observations to use and what products derived from those observations are available). Plans are passed to the scheduler as directed acyclic graphs of algorithms linked together by temporary files representing the inputs and outputs of the programs. The current RVC planner is Simple Non-Linear Planner (SNLP) (McAllester & Rosenblitt 1991), chosen primarily because it is small and hence easily modified. SNLP is written in Lisp.

Scheduler - a system that accepts plans and binds resources (CPU cycles and disk space) to them so they can be executed. The scheduler maintains a resource availability database to help it do its job. The current RVC scheduler is descended from a prototype scheduler named Data Archiving and Distribution System (DADS) (White *et al.* 1996) done for EOSDIS by Honeywell; it models the temporal availability of resources using their Time Map Manager (TMM) temporal constraint system. TMM and DADS are written in Lisp.

Dispatcher - a system that polls the Scheduler for algorithms that are ready to execute, launches them on the resources they are bound to, monitors their progress and reports their status back to the Scheduler. The current RVC dispatcher is a Tcl/Tk application.

Resources - computers registered with the Scheduler which can be used to execute plans. Currently the RVC can use any system running Unix as a resource.

Mass Store - file systems associated with resource machines where the RVC stores processed products.

All the parts of the RVC communicate with each other by passing commands and responses structured as S-expressions. S-expressions are easily generated and parsed by the different programming environments within the RVC (Lisp, C, C++, Tcl/Tk), and have enough structure to easily encode complex messages.

Example Product Generation A typical product generation within the RVC starts with a message like this, which is a request from an Access Client to generate a product of type `AVHRR1` starting from any products available from observation 350, and deliver the result to an anonymous FTP site on `my-host`:

```
(PROCESS
  (OID 350)
  (GOALS
    (GOAL (TYPE "AVHRR1") (FTP "my-host"))))
```

The Database takes the original message, adds in what it knows about observation 350, and forwards the augmented message to the PSD:

```
(PROCESS
  (ELEMENT (OID 350)
```

```
  (ATTRIBUTES
    (PLATFORM "NOAA-12") (SENSOR "AVHRR")
    (FILE "file://foo/mass/Q19971641043.L0_HRPT")
    (DATE (YEAR 1997) (MONTH 6) (DAY 13)
      (HOUR 12) (MINUTE 8) (SECOND 32))
    (ROWS 4497) (COLUMNS 2048)
    (BITS_PER_PIXEL 10)
    (STORED_BITS_PER_PIXEL 16))
  (EXISTING_TYPES
    (ALGORITHM (NAME "NOAA-12 AVHRR")
      (FILENAME
        "file://foo/mass/Q19971641043.L0_HRPT"))))
(PID 18)
(GOALS
  (GOAL (TYPE "AVHRR1") (FTP "my-host"))))
```

The PSD creates a plan to generate the requested product, allocates compute resources to execute it, and queues the resource-bound plan for execution, then informs the database that the product will be generated:

```
(MESSAGE (PID 18)
  (TYPE PROCESS) (STATUS SUCCESSFUL))
```

The PSD dispatches and monitors the processes of the plan as they run. When everything is finished, the PSD informs the database that the requested products are now available for anonymous FTP at the URLs in the FILES clause:

```
(MESSAGE (PID 18)
  (TYPE PROCESS_STATUS) (STATUS SUCCESSFUL)
  (FILES
    "ftp://my-host/pub/NOAA12_AVHRR1.gz"))
```

The database then e-mails the URLs to the requesting user.

How the PSD Works

Resource Registration The PSD views its domain as sets of *resources* and *algorithms*. Resources and algorithms are made available to the PSD by *registering* them: a curator clients sends the PSD an S-expression describing the new capability, the PSD unrolls it into an internal model, and then goes out across the network to gather anything else it needs to use the resource (probe storage sizes) or algorithm (copy and store executables).

A *resource* is a system the PSD may use to store files and run compatible programs. Resources are actually composed of a CPU type and one or more disk storage types. Here is an S-expression used to register a typical resource:

```
(NEW_RESOURCE (OID 113)
  (ETHERNET (HOST_NAME DANVILLE.GSFC.NASA.GOV)
    (IP_ADDRESS "202.170.170.8"))
  (FDDI (HOST_NAME DANVILLE-F.GSFC.NASA.GOV)
    (IP_ADDRESS "202.170.170.168"))
  (ARCHITECTURE
    (MAKE "HP") (MODEL "9000/735"))
  (OPERATING_SYSTEM
    (TYPE "HP-UX") (VERSION "B.10.10"))
  (ANONYMOUS_FTP (FTP_DIRECTORY "/pub"))
  (DATA_DIRECTORIES
    (DIRECTORY (NAME "/lost/rvc_online")
      (FILE_SYSTEM_TYPE ONLINE))
    (DIRECTORY (NAME "/mnt4/rvc_nearline")
      (FILE_SYSTEM_TYPE NEARLINE)))
  (EXECUTABLES_DIRECTORY "/usr/local/rvc/bin2"))
```

This description has one of everything the PSD considers to be a resource: a CPU (ARCHITECTURE and OPERATING_SYSTEM clauses), an area for anonymous FTP (used to make products available to the outside world), disk space for scratch storage while running programs (the ONLINE DATA_DIRECTORY) and disk space for long-term product storage (the NEARLINE DATA_DIRECTORY). The PSD takes this resource description and generates from it four scheduler resource descriptions (see below for examples).

```
(SCRIPT-ADDRESOURCE
  :PRETTY-NAME "danville.gsfc.nasa.gov"
  :NAME (COMPUTATION HP DANVILLE.GSFC.NASA.GOV)
  :ATTRIBUTES ()
  :AVAILABILITIES ((0 3616053101 1)))

(SCRIPT-ADDRESOURCE
  :PRETTY-NAME "danville.gsfc.nasa.gov:/scratch"
  :NAME (STORAGE DISK FILE-SPACE DSK22744)
  :ATTRIBUTES (DANVILLE.GSFC.NASA.GOV)
  :AVAILABILITIES ((0 3616053101 1997022)))
```

Each scheduler resource has a NAME in a hierarchical type space, a set of ATTRIBUTES in addition to its name, and a set of AVAILABILITIES describing how much of the resource can be used and at what times as a list of (*start-time end-time quantity*) specifications.

CPUs are named as (COMPUTATION *cpu-type domain-name*). Since the original resource description didn't specify an availability, the PSD assumes the resource is "always" available, from zero seconds to the scheduler's horizon, arbitrarily years from now. CPUs have one unit; as resources they are either free or busy. This model is simplistic, but not too bad given the nature of most image-processing code.

Disk storage is named as (STORAGE DISK *purpose unique-tag*). The domain names of any machine(s) that can access the storage are placed on the ATTRIBUTES list. Disk availability is measured in 1K byte blocks.

Resources are requested by specifying a (possibly incomplete) name, list of attributes, quantity, and duration; the scheduler then matches the specification against the available resources to come up with candidates. For example, a request for any HP CPU would have (COMPUTATION HP) as its NAME field.

Algorithm Registration and Planning An *algorithm* to the PSD is an executable program that takes typed input files and transforms them into typed out-

put files. Algorithms may have parameters and ancillary files. Here is an S-expression used to register a typical algorithm:

```
(NEW_ALGORITHM
  (ALGORITHM "navTIROS12")
  (ARCHITECTURE
    (MAKE "HP") (MODEL "9000/770"))
  (OPERATING_SYSTEM
    (TYPE "HP-UX") (VERSION "B.10.01"))
  (EXECUTABLE_FILENAME
    "exec://danville/mass/HP/RDCnavTIROS")
  (EXECUTABLE_SIZE 107108)
  (AUTHOR "Allen Lunsford (Hockey Legend)")
  (ARGUMENTS
    (ARGUMENT (USAGE INPUT_FILE)
      (DATA_TYPE "TIROSAUX12")
      (SWITCH "-iAUX")
      (PARAMETER "TIROSAUX12"))
    (ARGUMENT (USAGE INPUT_FILE)
      (DATA_TYPE "EPHEM")
      (SWITCH "-iEPHEM")
      (PARAMETER "EPHEM"))
    (ARGUMENT (USAGE OUTPUT_FILE)
      (DATA_TYPE "TIROSDROPOUT")
      (SWITCH "-oDROPOUT")
      (PURPOSE PRODUCT_GENERATION))
    (ARGUMENT (USAGE OUTPUT_FILE)
      (DATA_TYPE "GRID")
      (SWITCH "-oGRID")
      (PURPOSE PRODUCT_GENERATION))))
```

This description tells the PSD everything it needs to know to use this program:

- the name and location of the executable in the mass store if it needs to install it on a resource machine before running it (`EXECUTABLE_FILENAME` clause).
- the type of computer it runs on (`ARCHITECTURE` and `OPERATING_SYSTEM` clauses, identical to the ones in the resource description above).
- the number, order, and type of any arguments it takes (`ARGUMENT` clauses). The current PSD file typing system is deliberately simple: file types are symbolic tags in a flat type space. This turns out to be just enough information for a planner to be able to chain algorithms together.

The PSD takes this algorithm description and generates from it a SNLP plan step:

```
(DEFSTEP
  :PRECOND
    '((DATA-TYPE ?OID TIROSAUX12 ?I1120)
      (DATA-TYPE ?OID EPHEM ?I1130))
  :ACTION
    '("exec://danville/mass/HP/RDCnavTIROS"
      "-iAUX" ?I1120 "-iEPHEM" ?I1130
      "-oDROPOUT" ?O1140 "-oGRID" ?O1150)
  :ADD
    '((DATA-TYPE ?OID TIROSDROPOUT ?O1140)
      (DATA-TYPE ?OID GRID ?O1150)))
```

In the plan step, the `?I` variables represent input file names and the `?O` variables represent output file names; the `?OID` variable holds the object ID of an observation. The SNLP database fact (`DATA-TYPE` *observation-id file-type file-name*) means that *file-name* contains data of *file-type* derived from *observation-id*; thus the plan step says that given files of types `TIROSAUX12` and `EPHEM`, executing `RDCnavTIROS` will create files of types `TIROSDROPOUT` and `GRID`.

In addition to the plan steps generated from registered algorithms, the PSD has several hand-coded plan steps to handle moving typed files into and out of the mass store, and delivering them to anonymous FTP sites so users can retrieve them. Here, for example, is the plan step used to fetch a typed file from the mass store:

```
(DEFSTEP
  :PRECOND
    '((ON-SERVER ?OID ?TYPE ?CFILENAME))
  :ACTION
    '("gunzip-if-needed.script"
        -I ?CFILENAME -O ?FILENAME)
  :ADD
    '((DATA-TYPE ?OID ?TYPE ?FILENAME)))
```

And here is the step used to deliver a typed file to a FTP site:

```
(DEFSTEP
  :PRECOND
    '((DATA-TYPE ?OID ?TYPE ?CFILENAME))
  :ACTION
    '("ftptourl.script" ?CFILENAME ?FTPDEST)
  :ADD
    '((FTP-RESULT ?OID ?TYPE ?FTPDEST)))
```

`gunzip-if-needed.script` and `ftptourl.script` are shell scripts that can perform their function on any Unix machine; they need to be installed before a machine is registered with the PSD as a resource.

We now have enough plan steps to build complete plans to generate image products and return them to users. The PSD generates the planner initial conditions from a given observation *oid* by querying the database for the existing product types it has in the mass store for it; each of those types becomes a fact of the form:

(`ON-SERVER` *oid file-type filename*).

A user request for a product of type *file-type* derived from *oid* turns into the goal clause:

(`FTP-RESULT` *oid file-type* "`ftp://`*host/path/filename*")

Handing the initial conditions and goals over to SNLP allows it to generate a DAG of plan steps.

Scheduling and Resource Allocation After SNLP generates its plan, the scheduler takes over and allocates resources to execute it. For each step in the plan, a scheduler task description is generated. Scheduler tasks can specify their minimum and maximum duration, their earliest and latest permissible start and

finish times, their relationship to other tasks, and the amount and types of resources needed while they execute. Here is a typical subtask description for the primitive PSD operation of fetching a file from the mass store:

```
(SCRIPT-ADDSUBTASK
  :OID 141486
  :TASK 14148
  :INVOCATION
    "exec://danville/gunzip-if-needed.script
      -i file://danvillev/incoming/Q19971641043
      -o /scratch/TASK-14148/TEMPFILE-17471"
  :PREDS (141481)
  :SUCCS (141487)
  :MIN-DUR 1
  :MAX-DUR 600
  :RESOURCE-REQUIREMENTS
    ((1823745
       (COMPUTATION)
         NIL 1 :UNIQUE)
     (1823746
       (STORAGE DISK FILE-SPACE)
         NIL 100000 :BEGINS)))
```

Note particularly the `RESOURCE-REQUIREMENTS`, which is a list of (*id-number resource-name resource-attributes quantity temporal-scope*) specifications. Since this task is a shell script, it can run on any computer (if it needed an HP machine, it would have asked for `(COMPUTATION HP)`), and it needs the computer only for the duration of the task (the `:UNIQUE` tag). It moves a 100 MB file from the mass store to local disk, and the file stays there for future tasks to use (the `:BEGINS` tag; the task that deletes this file will have a specification with a `:ENDS` tag to free the resource).

Once all the plan steps have scheduler tasks associated with them, the scheduler is turned loose, and it tries to match the resource requirements with the resources it has registered. If it succeeds, the PSD can walk the resulting resource specifications and use them to fill in the blanks in the plan, mainly generating temporary directory and file names for intermediate results. At this point, any tasks with no predecessors are marked in the temporal database as ready-to-run.

Dispatching and Execution The PSD dispatcher is a Tcl/Tk application that polls the scheduler at fixed intervals (ten seconds or so), telling it to update its clock and asking it if anything new is ready-to-run. Tasks that are ready are launched on the resources bound to them (via remote shell call in most cases) with a small Tcl/Tk process monitoring them as they run. This execution monitor process sends messages to the scheduler to tell it when tasks start, when they finish, and whether they completed successfully or not (judged solely on their exit status code; registered algorithms must report their status correctly or risk confusing the scheduler).

When a task completes successfully, its successor tasks are marked as ready-to-run, and will be launched when the dispatcher next polls for them. When all the tasks of a plan complete successfully, the scheduler notices and sends a `PROCESS COMPLETE` status message. If a task executes for longer than its `MAX-DUR`, the scheduler will notice when the dispatcher updates its clock; it can then report the error.

Experience with Current System

The current fielded RVC system was created in rapid prototyping mode by integrating several prototype systems that have been under development in Code 935 since 1989. Our goal was to get a working system into the hands of real users as quickly as possible, to get feedback from them to guide future development. As could be expected with any prototype, we had to make design compromises and work around limitations in our tools to get the system out the door. We discuss some of those problems and our solutions to them below.

Planning SNLP turned out to be a useable tool for the simple plans we had to generate for RVC product generation. SNLP itself accounts for about half of the PSD's cycles. Much of this is due to the fact that SNLP is implemented for students to study and experiment with; it is coded for clarity more than high efficiency.

In particular, SNLP will sometimes run off and do much more work than necessary to generate plans, and there is no easy way to make it more efficient. For example, given a goal set asking for N file types and a plan step that had those types in its outputs, SNLP will generate N candidate plans, one for each output file. These plans look equally good to SNLP, so it will keep them all, and generate $N \times (N-1)$ plans at the next stage, ultimately ending up with $N!$ identical plans.

Scheduling The DADS-based scheduler performs quite impressively. The Honeywell TMM core algorithms are very efficient; scheduler resource allocation takes up at most one fourth of the PSD's cycles - about five seconds overhead on a typical twenty minute image product job.

The biggest problem we had with the scheduler was that we used it in a multi-threaded Lisp environment, and it wasn't designed with that in mind. The problem was that asynchronous events (polling from the dispatcher, status reports from the execution monitor) needed to access and update the resource temporal database, and there was no locking on that data structure. Often what looked from the outside like simple requests, like asking a task what its earliest start time was, turned out to require exclusive access to the temporal database. The PSD was failing intermittently for months until we realized this, and only became stable after we set up a process-lock and made scheduler-using tasks acquire it before doing anything dangerous.

Costs and Benefits As we started out, there was some worry that full-blown AI-based planning and

scheduling would be too expensive computationally - that the PSD would consume a significant fraction of a resource machine's memory and cycles for little benefit. This has turned out not to be the case.

The cost of having planning and scheduling in the RVC is small relative to the cost of running a typical image processing job. The Allegro Common Lisp executable containing the PSD is the largest single program in the RVC system, but it has turned out to be a well-behaved application and a good neighbor as far as system paging is concerned, even without any serious tuning. The system takes about twenty seconds to plan, schedule, dispatch and monitor a typical process request generating a basic set of image products from a TIROS or GOES data file; running the entire process takes about twenty minutes, so the overhead of the PSD is not significant.

The benefits of having planning and scheduling in the RVC are currently not as large as they could be, but are expected to grow as the system matures and is used to solve larger problems. The planner and scheduler are both underutilized in the current system.

The planner currently solves problems that are mostly within the capabilities of simpler tools like the Unix `make` utility. The main things the RVC gets from using a planner now are expressiveness and tolerance of ambiguity: it is safe to register algorithms that allow more than one way to create a given type of file, because the planner can be relied on to choose the one that is best according to its search strategy. The current system relies on this internally. When a product derived from a given observation is requested, the system can deliver it to an FTP site for pickup in several different ways:

- If the product has never been requested before, it can run algorithms to generate it from precursors in the mass store, then copy it to an FTP site.
- If the product is already in the mass store, it can copy it directly from there to an FTP site.
- If the product is already at an FTP site, it can just tell the user where it is.

The planner chooses an appropriate delivery method by selecting the plan with the fewest steps that works given the initial conditions for the observation.

The scheduler currently uses very coarse estimates for algorithm time and space utilization; this means the scheduler's estimates for how long a plan will take to run or how much disk space it will need to finish are educated guesses at best.

Future Plans

The rapid prototyping approach taken to develop the RVC system appears to have succeeded. The first release of the prototype RVC system was designed and implemented in about nine months. A full-fledged implementation, incorporating feedback from the original RVC sites as well as other improvements, should be completed and deployed at the various RVCs by the middle of 1998.

A number of enhancements are slated for the planner/scheduler module. Both the planner and scheduler could do more in the RVC given more information about their domains. The planner could be used to generate plans from fuzzier specifications of output requirements, if it had a better model of what algorithms do. The scheduler could generate more accurate estimates of resource utilization if it had a better model of how algorithms use resources.

One of the most significant (and most difficult) improvements will be in the addition of a hierarchical typing system throughout the IIFS, replacing and extending the current simple symbolic typing scheme. This will require each output type to produce an accompanying metadata file which will be maintained by the database. The planner will be able to pass specific attributes of this metadata to algorithms further along in the chain when the product is part of an intermediate step in producing a goal.

We are also cleaning up the PSD internally, adding abstract interfaces to planning, scheduling, and dispatching, to make it easier in the future to change planning or scheduling tools if needed. We already intend to try a different planner in the next major release, Lansky's action-based planner COLLAGE (Lansky 1994).

We also need to improve the execution monitor, so it can gather better resource utilization data. At present, the scheduler is under-utilized in the PSD, primarily because we currently can't give it good data about algorithm resource use. Having this data will be critical for one of our long-range goals, which is to allow RVC sites to share resources and cooperate in processing data.

The biggest hurdle in making these enhancements will be keeping them intelligible to the expected RVC user community. We expect to have around sixteen RVC systems out in the field by the time of the next major release; the users will be experts in remote sensing, and must not be required to become experts in computer science or AI to use their RVC.

References

Cromp, R. F.; Campbell, W. J.; and Short, Jr., N. M. 1993. An intelligent information fusion system for handling the archiving and querying of terabyte–sized spatial databases. In *International Space Year Conference on Earth and Space Science Information Systems*, 586–597. American Institute of Physics.

Lansky, A. 1994. Action-based planning. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*. American Association for Artificial Intelligence.

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proceedings of the Ninth Na-*

tional Conference on Artificial Intelligence, 634–639. American Association for Artificial Intelligence.

Short, N. M.; Cromp, R. F.; Campbell, W. J.; Tilton, J. C.; LeMoigne, J. L.; Fekete, G.; and Netanyahu, N. S. 1995. Mission to Planet Earth: AI views the world. *IEEE Expert* 24–34.

White, J.; Boddy, M.; Nelson, K.; Atkins, R.; and Bedet, J. 1996. Programmers guide to SWAMPANGEL scheduling. Technical Report software version v3-02, Honeywell Technology Center.

Makespan Scheduling for Assembly Tasks: Extended Abstract

Brian Drabble, David Clements
Computational Intelligence Research Laboratory,
University of Oregon,
1269 University of Oregon,
OR 97403-1269, USA.
Email: drabble@cirl.uoregon.edu, clements@cirl.uoregon.edu

Introduction

A number of applications in the space domain require complex assembly planning and scheduling. For example, assembly, integration and verification, satellite command sequencing and satellite assembly. One of the main objectives in each of these applications is to identify a minimum length schedule or makespan which accomplishs all the activities in the task. This is often difficult to achieve due to the complex interaction of resource and temporal constraints, e.g. variable shift patterns. The aim of this article is to provide a description of some of the techniques being developed at CIRL to develop schedules with minimum makespan. The techniques described in this article are Limited Discrepancy Search (LDS) (?) and the Doubleback Optimizer (?). The current scheduling system is acting as a test bed for further developments including a probablistic version of LDS and the ability to handle more complex concepts such as release times and delivery dates. The remainder of the article provides details of the techniques and describes the ways in which these could be used for applications such as assembly, integration and verification.

Scheduling Technologies

Overview

CIRL's scheduler is a research prototype designed to solve a particular class of problem. In particular, it attempts to generate minimum length, precedence and resource constrained job shop schedules. These problems typically involve manufacturing small numbers of fairly expensive items. To date the scheduler has been tested on a number of problems from airplane assembly manufacturing. The results obtained from these problems have shown that the CIRL scheduler is currently the best solution and manages to find schedules which are between one and three days better that other systems. This can be a significant saving when the costs of one days additional production vary between $100,000 and $1,000,000. The study of the generic features of the aircraft assembly problem has identified a number of similar tasks in the space domain. For example, in assembly, integration and verification (AIV) the aims are to integrate the different AIV schedules for the subsystems of the satellite into a single AIV schedule and minimize the overall cost of the combined schedule. By minimizing the length of the schedule the cost of using expensive test resources e.g. trollies, chambers, etc, is reduced. The AIV task has been shown to be a successful application for AI planning and scheduling technologies (?) with a deployed system for The European Space Agency being routinely used for payload checkout on the Ariane IV program. An additional application which would benefit from minimising makespan would be satellite imagery in which there are a large number of requests and limited time and resources.

Core Technology

The core technology of the CIRL scheduler is based around two different techniques to generate schedules. These are:

- The use of heuristics to attempt to generate a reasonably short "seed" schedule.
- The "seed" schedule produced by the heuristics is then fed to an optimizer that uses Doubleback optimization to shorten the schedule further.

The hueristic search strategy Limited Discrepancy Search (LDS) is used to generate good "seed" schedules using the heuristics, and does so in a manner that ensures that the schedules are not merely minor variants of each other. Work with the scheduler has discovered that by feeding multiple, different schedules to the optimizer that the scheduler can generally produce a better schedule than than can be obtained by feeding the single schedule that was derived using purely the heuristics.

Doubleback Optimization

Doubleback optimization, also known as schedule packing or bin packing, involves "sloshing" a candidate schedule, repeatedly, right and left within a scheduling window. This has a remarkable impact on the length of most schedules. The Doubleback process is analogous to filling a box with blocks and then shaking the box. Shaking the box will almost always result in a denser packing of blocks. Likewise, in schedules, Doubleback almost always results in a denser packing of tasks in a schedule.

Limited Discrepancy Search (LDS) and Heuristics

When generating a candidate schedule, many, many decision points are reached along the way about what task to place when. In scheduling, heuristics are used to attempt to predict which choices will result in the best (i.e. shortest) schedule. However, there are no known perfect scheduling heuristics. A perfect heuristic, if one existed, would enable the scheduler to always place tasks at the right time. An optimal schedule could be generated every time. LDS helps make up for the absence of a perfect heuristic, particularly in the case where you have a good heuristic. (In fact, the better your heuristic, the better LDS works.)

In scheduling problems of any size it is unlikely that always using a merely good heuristic will get you really close to an optimal schedule. A merely good heuristic will be incorrect some of the time. As the complexity of scheduling problems increases, the number of decisions guided by the heuristic also increases. The more decisions made, the more likely it is that some of them are going to be incorrect. How does LDS help address this problem? LDS is a systematic method for disregarding the recommendation of a heuristic a *limited* number of times (thus the name LDS) when generating a schedule. With LDS, schedules are generated repeatedly, each of them following the heuristic for all decisions except one. The decision at which the heuristic is ignored is different in each schedule.

If the heuristic leads to only one incorrect decision, then using LDS1 (the fastest form of LDS) will lead to a perfect schedule. Even if the heuristic leads to more than one incorrect decision (which is usually the case) then LDS1 will likely lead to a better solution then always following the heuristic.

Features of the CIRL Scheduler

The current CIRL scheduler currently supports a number of features and these are as follows:

- Fixed duration tasks with fixed resource usage for the duration of the task.
- Resource availability that is constant over time, or that varies cyclically over time, e.g. shifts
- Allocatable (i.e. non-consumable resources).
- Precedence constraints between tasks that state "this task cannot start until this other task has finished". An arbitrary network of precedences can be specified as long as there are no cycles present.

- A general constraint that all tasks running at each particular time must use no more than the amount of resources that are available at that time.

Research efforts are continuing to develop the functionality of the scheduler and these include using probablistic measures in seed generation and including release times and due dates in the schedulers representation.

References

Aarup, M. , Arentoft, M.M., Parrod, Y., Stokes, I., Vadon, H. and Stader, J. (1994) Optimum-AIV: A Knowledge-Based Planning and Scheduling System for Spacecraft AIV, in Intelligent Scheduling (eds. Zweben, M. and Fox, M.S.), pp. 451-469, Morgan Kaufmann.

Crawford, J., An Approach to Resource Constrained Project Scheduling, in the proceedings of the Artificial Intelligence and Manufacturing Research Planning Workshop, Alberquerque, New Mexico, June 24-26, 1996. Also available in `http://www.cirl.uoregon.edu/crawford/papers/albur.ps`

Harvey, W, and Ginsberg, M., Limited Discrepancy Search, in the proceedings of IJCAI-95, Montreal, Canada, July, 1995, Also available in `ftp://www.cirl.uoregon.edu/pub/users/ginsberg/papers/lds.ps.gz`.

Acknowledgements

Effort sponsored by the Defense Advanced Research Projects Agency (DARPA) and Rome Laboratory, Air Force Materiel Command, USAF, under agreements numbered F30602-95-1-0023 and F30602-97-1-0294. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The work reported here was developed by a number of staff and students at CIRL and includes: Tania Bedrax-Weiss, Andrew Parkes, Bart Massey, Matt Ginsberg, David Joslin, David Etherington, Dave Clements, Jimi Crawford and others.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, Rome Laboratory, or the U.S. Government.

Repairing Plans On-the-fly

Brian Drabble, Jeff Dalton & Austin Tate
Artificial Intelligence Applications Institute
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
United Kingdom
Tel: (+44) 131 650 2732 Fax: (+44) 131 650 6513
E-mail: drabble@cirl.uoregon.edu, j.dalton@ed.ac.uk & a.tate@ed.ac.uk

Abstract

Even with the most careful advance preparation, and even with inbuilt allowance for some degree of contingency, plans need to be altered to take into account execution circumstances and changes of requirements. We have developed methods for repairing plans to account for execution failures and changes in the execution situation. We first developed these methods for the Optimum-AIV planner designed to support spacecraft assembly, integration and verification at ESA, and later deployed for Ariane IV payload bay AIV. This system was itself based on our Nonlin and O-Plan planning algorithms and plan representation. We subsequently refined the methods for the O-Plan planner and incorporated plan repair methods into the system. This paper describes the algorithms used for plan repair in O-Plan and gives an example of their use.[1]

Introduction

Even with the most careful advance preparation and even allowing for some degree of contingency pre-built into the plans, any plan being executed in the real world will have to be adapted to take into account execution circumstances and changes of requirements. For example, a deep space probe may require to adapt to new science experiments as new information leads to further experiments. Alternatively, cases such as Galileo have shown that failures in the spacecraft's hardware may need to be overcome by altering the current set of tasks and plans.

One of the aims of the O-Plan project (Currie and Tate, 1991; Tate et. al. 1994b; Tate et. al. 1996) during Phase II of the DARPA/Rome Laboratory Planning Initiative (Tate, 1996a) was to develop techniques to allow plans to be changed to take into account modifications in the task requirements and in the execution environment. The techniques allowed a failure to be identified and repaired with minimum impact on the rest of the plan.

The basis for the techniques was first developed for the Optimum-AIV planner (Aarup et. al. 1995; Tate, 1996b) designed for spacecraft assembly, integration and verification support at ESA and later deployed for Ariane IV payload bay AIV.

This paper will briefly describe some of the background work on O-Plan and Optimum-AIV, and then describe the algorithms used for plan repair in O-Plan. The paper describes a demonstration which was conducted in a command, planning, and control environment of the US air force. The task was to evacuate a number of foreign nationals from the fictional island of Pacifica (Reece et. al. 1993) and to transport them to safety. While the example is not directly related to the space domain, the demonstration does show how new requirements and changes in the environment can be integrated into an ongoing and executing plan and would be of use in the solving problems such as AIV, control of autonomous spacecraft, and lander missions.

[1] Brian Drabble is now a member of the Computational Intelligence Research Laboratory, University of Oregon.

Optimum-AIV – Assembly, Integration and Verification Planning

Planning is a key issue in the management of the assembly, integration and verification (AIV) activities of a space project. Not only must technological requirements be met, but cost and time are critical. There are costly testing facilities which must be shared with other projects, and there is a need to plan the coordination between a number of participants (agencies, contractors, launcher authorities, users). A delay caused by one participant normally leads to serious problems for others. Managers at all levels of a space project are concerned with planning, and they control closely the progress of the work. However, it has been difficult to find computer-based planning aids which meet the needs of this application. General purpose project management software cannot represent the wide range of factors to be taken into account, and is too complex to be used to interactively modify plans during project execution (Parrod et. al. 1993). For this reason, the European Space Agency commissioned the Optimum-AIV system which utilizes AI planning representations and techniques (Aarup et. al. 1995; Tate, 1996b).

The system which was developed was based on the earlier Nonlin (Tate, 1977) and O-Plan (Currie and Tate, 1991; Tate et. al., 1994b) planning algorithms and plan representation. The following techniques are used in Optimum-AIV:

- Optimum-AIV adopts a partially-ordered plan representation, which supports causally independent activities that can be executed concurrently.
- It searches through a space of partial plans, modifying them until a valid plan/schedule is found.
- The system employs hierarchical planning. The term hierarchical refers to both the representation of the plan at different levels, and also the control of the planning process at progressively more detailed levels.
- During plan specification and generation, the system operates on explicit preconditions and effects of activities that specify the applicability and purpose of the activity within the plans. With this knowledge, it is possible to check whether the current structure of the plan introduces any conflicts between actual spacecraft system states, computed by the system, and activity preconditions, which have been specified by the user. Such conflicts would arise if one activity deletes the effect of another, thus removing its contribution to the success of a further activity. The facility for checking the consistency of the plan logic, by dependency recording, is not possible within existing project management tools, which assume that the user must get this right.
- Detailed constraints are associated with the plan. These represent resource and temporal constraints on the activities in the plan as well as a more general class of global activity constraints. The scheduling task in Optimum-AIV is considered as a constraint satisfaction problem solved by constraint-based reasoning. The constraints are propagated throughout the plan, gradually transforming it into a realizable schedule. Invariably not all of the constraints can be met, such that some have to be relaxed via user intervention.
- During planning, the system records the rationale behind the plan structure; that is, user decisions on alternatives are registered. This is used to assist during plan repair where the user tries to restore consistency. Information can then be derived about alternative activities, soft constraints that may be relaxed, and potential activities that may be performed in advance.
- Test Failure Recovery Plans are available as plan fixes to enable the plan to be brought back on track after the failure of a test during the assembly and integration process. The same AI planning methods used to generate a plan are also used to assist in fixing such problems. Optimum-AIV assists the user in plan repair in an interactive way rather than performing the

repair itself.

Following an evaluation of Optimum-AIV at ESA, it has been reported (Parrod et. al, 1993) that the system is in use for planning the production of the vehicle equipment bays of the European Ariane IV launcher. It was reported that the system was chosen by the Araine IV project team due to the following:

- the wealth of information which can be provided to and used by the tool to describe the constraints inherent in the AIV activity.
- the quality of support provided by the tool to allow resource conflicts to be resolved.
- the clear representation of information and the interactive capabilities which enables engineering management to access several planning scenarios on-line.
- the fact that Optimum-AIV provides a single solution to problems of managing the plan, schedule, and allocation of resources amongst competing vehicle equipment bays which are concurrently being assembled.

Optimum-AIV provides a rich plan representation and aids to allow for the editing of AIV planning information and a wide range of constraints on the process. This information forms a basis for plan generation, checking of plan logic, and analysis of plans. Facilities are available to allow for the interactive repair of executing plans when tests indicate failures of components under assembly and integration. Optimum-AIV is an example of a deployed application of a number of AI planning techniques.

O-Plan Demonstration and Scenario Description

A demonstration experiment was performed which showed O-Plan (Tate et. al., 1996) solving a number of tasks from an integrated command, planning and control scenario related to the performance of Non-combatant Evacuation Operations (NEOs) on the fictional island of Pacifica (Reece et. al. 1993). The aims of the demonstration were to show:

- O-Plan reacting to changes in the environment and identifying those parts of the plan which were now threatened by these changes.
- O-Plan reacting to changes in the overall task by integrating new plan requirements into the plan.

The types of plan repairs explored in this demonstration include responses to failures of trucks due to blown engines and tyres and the inclusion of new task objectives, such as to pick up an extra group of evacuees. The Pacifica scenario used for the demonstration is a simplification of a real logistics problem of interest to the DARPA/Rome Laboratory Planning Initiative (Tate, 1996a). The plan schema library for this domain contained 12 schemas which defined alternative evacuation methods: trucks or helicopters, fuel supplies, transport aircraft, etc. The plans generated contained an average of 20 actions and were developed in approximately 40-60 seconds. Four different repair plans were used in the demonstration:

- Three cases repairing a broken engine on a ground transport:
  - The engine can only be fixed by a repair crew which is dispatched from the Pacifica airport at Delta with a tow truck. The ground transport is then towed to Delta for repairs. The evacuees remain with the ground transport while it is being towed.
  - The failure of the transport occurs in a time critical situation and there is insufficient time to tow the broken transport to Delta. The evacuees are moved from the broken ground transport by helicopter to Delta and the transport is abandoned.
  - The failure of the transport occurs in a time critical situation, and the evacuees are moved by another ground transport instead of by helicopter.

- One case reparing a blown tyre on a ground transport:
  - The driver of the ground transport can fix the tyre by the side of the road. The effect of the repair action is to delay the ground transport by a fixed amount of time.

In addition, a closely allied Ph.D student project by Glen Reece developed a more comprehensive reactive execution agent called "REA" (Reece, 1994; Reece and Tate, 1994) based on the O-Plan architecture. It has been used to reactively modify plans in response to operational demands in a simulation of the Pacifica island in the context of a NEO.

O-Plan Plan Repair Algorithms

The plan-repair mechanisms allow O-Plan to integrate a number of pre-assembled repair plans—e.g., to repair a blown engine, or to repair a flat tyre—into an ongoing and executing plan. Although the integration was performed by the planning agent, the techniques and methods could easily have been added to the capabilities of a separate execution agent – as in Reece's REA.

O-Plan's internal plan representation contains two tables used by the plan repair algorithms to determine the consequences of failures: the **Table of Multiple Effects** (TOME) and the **Goal Structure Table** (GOST). Plans contain actions (nodes), and actions can have effects. Effects can take place at either end of an action: (`begin_of`) or (`end_of`). Each effect is recorded in the TOME by an entry of the form ⟨pattern⟩ = ⟨value⟩ at ⟨node-end⟩. For example, `(colour_of ball) = green at end_of node-1`.

When an action depends on an effect asserted earlier, that is recorded in the GOST by an entry of the form ⟨condition-type⟩ ⟨pattern⟩ = ⟨value⟩ at ⟨condition-node-end⟩ `from` ⟨contributor-node-ends⟩. This specifies a protected range: ⟨pattern⟩ = ⟨value⟩ is asserted at one of the contributor-node-ends and is required at the condition-node-end. For example, `unsupervised (colour_of ball) = green at begin_of node-2 from (end_of node-1)`.

These tables are maintained by the O-Plan TOME and GOST Manager (TGM) - a plug in constraint manager in the O-Plan Architecture (Tate et. al. 1996). A plan repair is required when one or more of the GOST entries are broken—i.e. a contributor of a GOST entry is not asserted as expected, or an external world event occurs and has extra effects that break a protected range by undoing a required effect.

Plan repairs are dealt with by a number of knowledge sources—pieces of code which deal with a specific aspect of the planning problem. The knowledge sources are responsible for determining the consequences of unexpected events, or of actions that do not execute as intended, for deciding what action to take when a problem is detected, and for making repairs to the effected plan.

O-Plan maintains an agenda of "issues" that need to be resolved in the plan. For each type of issue, there is a corresponding issue handler (called a knowledge source in O-Plan). The top-level control structure in O-Plan is a loop that repeatedly selects an issue from the agenda and calls the appropriate knowledge source. When describing algorithms below, we will therefore sometimes speak of "posting" an agenda entry, where the issue type is represented by the knowledge source name (KS-CONTINUE-EXECUTION, KS-FIX, etc.)

The two types of problems that are dealt with by the repair mechanisms can now be described in more detail:

- **Execution Failure**:
  An execution failure occurs when one or more of the expected effects at a node-end fail to be asserted. For example, the node-end corresponding to the end of the action `Check_out_ground_transport` should assert that the status of the engine and tyres is fine: `(engine_status gt1) = working` and `(tyre_status gt1) = working`. This may not in fact be so if the action has not executed correctly. This type of failure may cause problems if the expected effects of the action are needed

to satisfy the preconditions of a later action. For example, the evacuation of people from an outlying city can only precede if the tyres and engine of the ground transport continue to function correctly.

- **Unexpected World Event:**
  Unexpected events cause effects in the world which can make planned actions fail. For example, a landslide event may have the effect `(road_status Abyss_to_Barnacle) = closed` and this would interfere with any action requiring the road to be open.

The description of the algorithms of the execution and plan repair system is divided into three main sections. The first describes how the system maintains an execution fringe of the node-ends awaiting execution; the second describes how the system deals with plan failures; and the third describes how it handles unexpected world events.

Further details of the algorithms and the demonstration experiments is given in Drabble et. al. (1995).

Maintaining the Execution Fringe and "Necking" the Plan

An activity is represented in an O-Plan plan as a node with two ends (time points). Conditions and effects can be attached to either end of a node and are monitored by the execution system. The system reasons purely in terms of conditions and effects at node-ends and not in terms of their associated activities or events[2].

The "execution fringe" is the list of node-ends currently ready for execution. A node-end is ready when all node-ends that must execute before it in the partially ordered plan have completed execution.[3] When ready, it can be dispatched for execution. That involves sending a message to an execution agent, which in turn sends messages to a world simulator. The simulator maintains a picture of the world in which execution is taking place, for demonstration purposes. As actions begin and end in the world, the demonstration simulator reports back to the execution agent, resulting in success and failure messages about the corresponding node-ends being sent from the execution system to the planner. When the planner receives a success or failure message about a node-end, it marks the end as having completed execution; and that may lead to further node-ends being considered ready.[4]

By keeping track of which node-ends have finished execution, the system maintains a context within which replanning for plan repair can take place and can establish a focus point when considering where to insert repair actions—after all node-ends which have executed and before any node-ends waiting to execute. This point is known as the plan's *neck point* and a single dummy node can be added to the plan by the repair algorithm to *neck* the plan at that point, when necessary.

Note that the "ready to execute" check for a node-end E considers only whether all the node-ends that must execute before E have been executed, regardless of whether the execution was successful. It assumes that any problems due to execution failures or world events have been fixed, and it is the responsibility of other parts of the system to ensure that this is so.

A node-end that is ready can have its status set back to not-ready after a plan repair, because the repair may introduce new actions that must execute first.

Dealing with Execution Failures

When an execution failure occurs at a particular node-end, some of the expected effects may not occur. They are returned from the execution monitoring system to the planning agent as a list of failed-effects. The task of the planning system is

[2]This allows plug in temporal constraint managers to be employed such as Tachyon (Stillman et. al., 1996) or TMM (Boddy, 1996).

[3]This check considers both links explicitly in the plan and temporal constraints maintained by a Time-Point Network Manager (TPNM) and other plug in constraint managers in O-Plan.

[4]It is assumed that execution is not so rapid relative the the planner's ability to respond that the planner's model becomes significantly out of date.

to fix the plan so that any condition that needed one of the failed effects as a contributor is satisfied in some other way. The fix can be relatively simple if there is already another contributor in the GOST entry or if there is a suitable alternative contributor already present in the plan. If these simple fixes cannot be applied, then the system will attempt to add a new action to the plan. However, if nothing requires the failed effects, then the execution "failure" can be ignored.

The main algorithm used by the system to track execution and initiate repairs is as follows:

- Mark the node-end as having been executed.
- If there are no failed effects, then a repair is not needed.
- If there are failed effects then remove the TOME entries that correspond to them.
- Determine which GOST entries are affected by the failed (removed) effects. If there are none, then a repair is not needed.
- At this point there is a failure that must be repaired.
  - Search through the affected GOST entries in turn. If a GOST entry has more than one contributor, check if any are still valid. If so, reduce the contributor list; otherwise record the GOST entry as truly broken.
  - If no GOST entries are truly broken, then the repair is complete.
- At this point, some GOST entries are truly broken and result in "issues" that must be resolved. For each of the broken GOST entries, post a KS-FIX agenda entry. When that agenda entry is processed, the KS-FIX knowledge source will be invoked, and it will consider two repair methods for satisfying the condition in the broken GOST entry:[5]
  - Find an existing alternative contributor in the plan.
  - Bring in additional actions (a repair plan) which assert the appropriate effect. Any new nodes will be linked after the *neck point* described above.
- Post a KS-CONTINUE-EXECUTION to continue execution after the fixes have been made.

Certain details of the repair depend on the type of the condition recorded in the broken GOST entry. In particular, a `supervised` condition (Tate et. al. 1994a) is unlike all other types because it requires that a ⟨pattern⟩ = ⟨value⟩ assignment be true *across a range*, rather than only at a single point.

Suppose a broken GOST entry g has the form `supervised` $p = v$ `at` e `from` (c). Then c is a node end that asserts $p = v$, and $p = v$ must be so not only at node-end e (which is all that other condition types would require) but also at node ends between c and e that are *spanned by the condition*. These are the siblings of c and e that are explicitly linked between c and e, or the descendants of such siblings, where two node-ends are siblings if they were introduced as sub-actions of the same action.

Broken `supervised` conditions are handled as follows:

- Create a new dummy node d to act as the "delivery point".
- Link d after the neck point, before e, and before all node-ends that are spanned by the condition and have not yet been executed.
- Change the GOST entry to list d as the contributing node-end, and give $p = v$ at d as an effect in the TOME.
- Post a KS-FIX to re-establish $p = v$ at d.

The system must be consistent in its use of the "ends" (`begin_of` and `end_of`) of d to avoid "gaps" in the goal structure which would effect the meaning of the plan.

[5] The "fix" issue introduces a condition of type `achieve` as described in (Tate et. al., 1994a).

Dealing with Unexpected World Events

When a significant event that is not in the plan occurs in the world, it is reported to the planner as a time, an event pattern, and a list of effects (of form ⟨pattern⟩ = ⟨value⟩). For instance, the occurrence of a landslide might be reported as:

```
event {landslide} with effects
   {status road-a} = blocked,
   {status road-b} = blocked;
```

Events are treated the same way as plan activities except they are not placed in the plan until they have occurred. The effects may break GOST ranges in the plan and if so, the planner must try to satisfy those conditions some other way. However, even if no GOST entries are broken, the planner needs to add a node to represent the world event. This is because, even if the event's effects don't make any difference now, they may matter later on.

The new event node represents something that has definitely and already happened. So it must be linked after all node-ends that have already been executed and before all node-ends that have not yet been executed.

The algorithm for dealing with unexpected world events is as follows:

- Add an event node, E, to represent the world event. Link it as described above. Mark E as having already been executed.
- Edit the GOST to remove any contributors that can no longer contribute, and get a list of the truly broken GOST entries. A contributor is removed when:
  - the condition is at a node-end that has not been executed,
  - the contributor is a node-end that has been executed, and
  - the unexpected world-event has a conflicting effect.
- For each truly broken GOST entry g, post a KS-FIX agenda entry as in the case of an execution failure, using end_of E as a neck point.
- Add the world event's effects at end_of E.
- If there were no truly broken GOST entries, then we are finished. Otherwise, Post a KS-CONTINUE-EXECUTION to continue execution after the fixes have been made. (The fixes will be made by processing the KS-FIX agenda entries.)

Conclusions

This paper has shown that current AI planning and scheduling techniques have reached the point where they can be deployed in real-world applications. This means real plan execution in the face of uncertainty and changing circumstances must be dealt with. Systems such as Optimum-AIV and O-Plan have shown that they provide valuable support to human users in identifying the point of failure in a plan and suggesting appropriate repairs. The techniques described in this paper to support plan repair are general enough to be applied in a wide variety of planning and scheduling applications.

Acknowledgements

The O-Plan project is sponsored by the Defense Advanced Research Projects Agency (DARPA) and Rome Laboratory, Air Force Materiel Command, USAF, under grant number F30602-95-1-0022. The O-Plan project is monitored by Dr. Northrup Fowler III at the USAF Rome Laboratory. The U.S. Government is authorised to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either express or implied, of DARPA, Rome Laboratory or the U.S. Government.

References

Aarup, M., Arentoft, M.M., Parrod, Y., Stokes, I., Vadon, H. and Stader, J. (1994) Optimum-AIV: A Knowledge-Based Planning and Scheduling System for Spacecraft AIV, in Intelligent Scheduling

(eds. Zweben, M. and Fox, M.S.), pp. 451-469, Morgan Kaufmann.

Boddy, M.S. (1996) Temporal Reasoning for Planning and Scheduling in Complex Domains: Lessons Learned, in Advanced Planning Technology, pp. 77-83, (Tate, A., ed.), AAAI Press.

Currie, K. and Tate, A. (1991) O-Plan: the Open Planning Architecture, Artificial Intelligence Vol. 52, pp. 49-86, Elsevier.

Drabble, B. (1995) Applying O-Plan to the NEO Scenarios, Appendix O in Tate, A., Drabble, B. and Dalton, J. (1995), An Engineer's Approach to the Application ofKnowledge-Based Planning and Scheduling Techniques to Logistics, Final Technical Report RL-TR-95-235, Rome Laboratory, Air Force Materiel Command, Rome, New York. Also available as DARPA-RL/O-Plan/TR/23 dated July 1995.

Parrod, Y., Valera, S. (1993) Optimum-AIV, A Planning Tool for Spacecraft AIV, in Preparing for the Future, Vol. 3, No. 3, pp. 7-9, European Space Agency.

Reece, G.A., (1994) Characterization and Design of Competent Rational Execution Agents for Use in Dynamic Environments, Ph.D Thesis, Department of Artificial Intelligence, University of Edinburgh, November 1994.

Reece, G.A. and Tate, A. (1994) Synthesizing Protection Monitors from Causal Structure, Proceedings of the Second International Conference on AI Planning Systems (AIPS-94), AAAI Press, Chicago, USA, June 1994.

Reece, G.A., Tate, A., Brown, D. and Hoffman, (1993) M., The PRECiS Environment, Paper presented at the ARPA-RL Planning Initiative Workshop at AAAI-93, Washington D.C., July 1993. Also available as University of Edinburgh, Artificial Intelligence Applications Institute Technical Report AIAI-TR-140.

Stillman, J., Arthur, R. and Farley, J. (1996) Temporal Reasoning for Mixed Initiative Planning, in Advanced Planning Technology, pp. 242-249, (Tate, A., ed.), AAAI Press.

Tate, A. (1977) Generating Project Networks, Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-77), pp. 888-893, Cambridge, MA, USA, Morgan Kaufmann.

Tate, A. (1996a) Advanced Planning Technology, AAAI Press.

Tate, A. (1996b) Responsive Planning and Scheduling Using AI Planning Techniques, Trends and Controversies, IEEE Expert - Intelligent Systems and Their Applications, Winter 1996.

Tate, A., Drabble, B. and Dalton, J. (1994a) The Use of Condition Types to Restrict Search in an AI Planner, Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94), pp. 1129-1134, Seattle, USA, August 1994.

Tate, A., Drabble, B. and Kirby, R. (1994b), O-Plan2: an Open Architecture for Command, Planning and Control, in Intelligent Scheduling, (eds, M.Zweben and M.S.Fox), Morgan Kaufmann.

Tate, A., Drabble, B. and Dalton, J. (1996), A Knowledge-Based Planner and its Application to Logistics, in Advanced Planning Technology, pp. 259-266, (Tate, A., ed.), AAAI Press.

Long Term Evolution of a Planning and Scheduling Capability for Real Planetary Applications

Principal Author
William "Curt" Eggemeyer (curt@eraserhead.jpl.nasa.gov)
Co-Authors
Sven U. Grenander (sven@sven.jpl.nasa.gov)
Stephen F. Peters (stevep@stevep.jpl.nasa.gov)
Arthur V. Amador (arthur.v.amador@jpl.nasa.gov)

Sequence Automation Research Group (SARG), Section 314
Jet Propulsion Laboratory, California Institute of Technology
Mail Stop 301-250D
4800 N. Oak Grove Dr.
Pasadena, Ca. 91109

Abstract

Plan-IT-II [1,2,3,4,5,6] represents a powerful capability for planning and scheduling of activities to be executed by interplanetary spacecraft. Its development and gradual acceptance by conservative flight projects spans 16 years after a couple of false in the prior 3 years. The development has been both evolutionary and revolutionary as lessons have been absorbed and major rewrites have been warranted. The application and adaptation of Plan-IT over a dozen mission domains has resulted in a representation which is rich enough to encompass virtually any interplanetary mission while at the same time allowing its use for ground-based manual, or fully autonomous on-board sequencing. Its application to fully autonomous sequencing domains has demonstrated that the next challenge in its development will be to provide user hooks which ease its use in developing and beta testing strategies or heuristics which will perform as desired under actual mission circumstances.

Introduction

This paper describes the various Plan-IT efforts we have been involved with over the years and concludes with some lessons learned and recommendations for future enhancements. Plan-IT was originally developed to "enhance" the performance of sequencing/scheduling personnel for spacecraft missions. Immediately, we became aware of the need to represent the problem to the user in a visual manner (Gantt chart style timeline) while minimizing the user's need to understand what the tool was displaying to them. Additionally, the tool needed to interact with users on their own terms, for example, "move this activity here," or "move this activity anywhere else," or "shrink this activity to fit between these activities," etc. In addition the tool needed to have the ability to be quickly adaptable while having the representational capacity to address complex sequencing problems. Finally, the tool needed to react quickly to user inputs.

SpaceLab, Space Station Power Scheduling Proof of Concept [7], Deep Space Network (DSN) Application [8], and the Comet Rendezvous Asteroid Flyby (CRAF) Demonstration [9] were all performed with the first major version of Plan-IT. The EOS demo [12,13], Galileo, Mars Pathfinder, Autonomous Nav-Sequencing, Distribution and Automation Technology Advancement of the Colorado Hitchhiker and Student Experiment of Solar Radiation (Data-Chaser), Microspacecraft, Distributed Object Interface demonstration, SIRTF demonstration, and the Cassini team personnel plan were done with the major revision of Plan-IT called Plan-IT-II. Currently, we are in the midst of adapting Plan-IT-II for New Millenium's Deep Space One Mission. The following descriptions of the various Plan-IT adaptations are in a general chronological order starting in the early 80's, although several of these efforts were done concurrently.

SpaceLab

The SpaceLab effort required Plan-IT to operate within an already existing scheduling system, called Experiment Scheduling Program (ESP). Plan-IT's task for this application was to give the user the ability to tweak an already existing schedule either graphically, by manual edits, or by algorithmic strategies specially coded for the SpaceLab problem domain. This task illustrated the usefulness of the timeline GUI for dealing with scheduling problems.

Space Station Power Scheduling Proof of Concept

The Space Station Power Scheduling demonstration was one of the first successes of Plan-IT's approach. This ap-

plication required Plan-IT to work with simple prioritized activities and real-time dynamic changes to update the schedule as changes occurred during its execution.

Deep Space Network (DSN) Application

Plan-IT was adapted in six months for scheduling the allocation of DSN antennas around the world. Plan-IT enhancements developed for this problem domain included easing the user edits, handling of generic as well as specific requests, and specialized algorithmic strategies. Plan-IT was used as an interim solution to the DSN scheduling problem until the Resource Allocation Planning Helper (RALPH) system development was completed. Plan-IT successfully demonstrated its capabilities by reducing the DSN turn-around time for scheduling by two orders of magnitude over existing manual scheduling methods.

Comet Rendezvous Asteroid Flyby (CRAF) Demonstration

This was the first successful demonstration of Plan-IT's application to deep space missions. Additionally, Plan-IT was combined with a natural language understanding system [10], enabling Plan-IT to take requests for the spacecraft in English form and translate them into activities which were then scheduled. This demonstrated to JPL management that such a "user-natural" scheduling system could make significant contributions to the spacecraft command and control process.

Earth Observing System (EOS)

Plan-IT was used to prototype two different types of nodes in the EOS distributed planning and scheduling system: the control center for the single complex instrument, and a remote science scheduling tool for that instrument. The prototypes were based upon science requests and specifications for the Advanced Spaceborne Thermal Emission Reflectance Radiometer (ASTER) instrument and for the AM1 platform.

For the ASTER Instrument Control Center (ICC) prototype, a scheduling algorithm incorporating nominal scheduling policy guidelines was encoded within Plan-IT. Interfaces were added which coordinated schedules with the EOS Control Center at the Goddard Space Flight Center, and accepted specific observation requests from a remote science workstation at the University of Colorado at Boulder. The ICC prototype was the only node in the system which provided full visibility into all constraints affecting, and conflicts involving, the ASTER instrument. A new averaging resource was added to Plan-IT to support modelling of mission guidelines.

For precise science planning, the ASTER science scheduling tool prototype modelled both "soft" science constraints (desires) and "hard" instrument and spacecraft constraints. A database of science requests and all target visibility opportunities was added to Plan-IT. A new request display showed the set of requests involving targets which could be observed over a specified interval of time. Using this display, the scientist could compare the requests competing for spacecraft resources over that interval. A new opportunities display showed data about specific observable targets over a time interval and provided an interface for subsetting these choices by telescope, pointing angle, original request, target, or individual opportunity. Using this interface, the scientist could assert the desire for specific observations in the scheduling timeline. Instrument activities satisfied instrument, spacecraft, and asserted science observation constraints. These prototypes contributed to the requirements for the EOS ground system.

Galileo (GLL)

In 1995, Plan-IT-II was adapted for the Galileo Mission to Jupiter as part of the redesign of both flight and ground systems to work around the partial deployment of the spacecraft's high gain antenna. The absence of a functional high gain antenna meant that the mission's data would have to be returned at significantly lower data rates. Flight software was redesigned to employ data editing, compression and buffering to deal with the high data rates produced by the spacecraft's instruments. In Figure 1 on the next page is Galileo's uplink data flow.

Plan-IT-II allowed science planners to model the data production rates of the various onboard instruments as well as the recording, editing, compression and buffering of the data by the spacecraft's main flight software. By using Plan-IT-II, users were able to adjust controls on the spacecraft's instruments and in the main onboard computer to match the production of data with the spacecraft's changing downlink bandwidth capability .

As shown on next page in Figure 2, Plan-IT-II was used to produce sequence planning inputs to the Uplink System's sequence generation software, Seqgen. By modeling the sequence used to command Galileo's tape recorder, Plan-IT-II produced a map of the data that was recorded on the tape recorder. This map was used to create a table of playback directives used by the onboard flight software to select and compress data from the tape recorder before placing it in the spacecraft's downlink buffer.

Plan-IT-II was also used to model the process of playing back encounter sequence data recorded on Galileo's tape

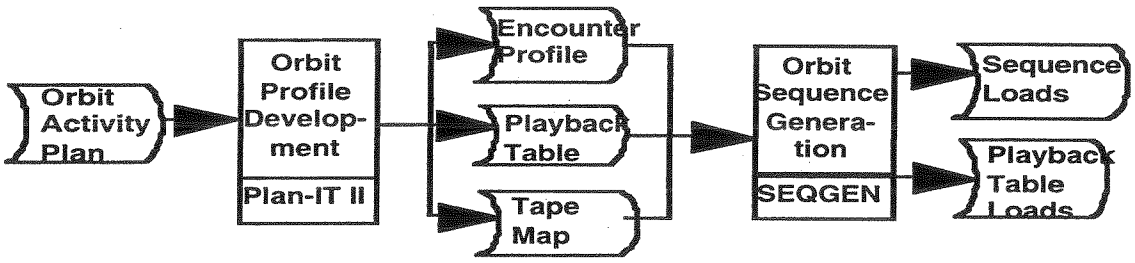


Figure 1: Galileo's Uplink Data Flow

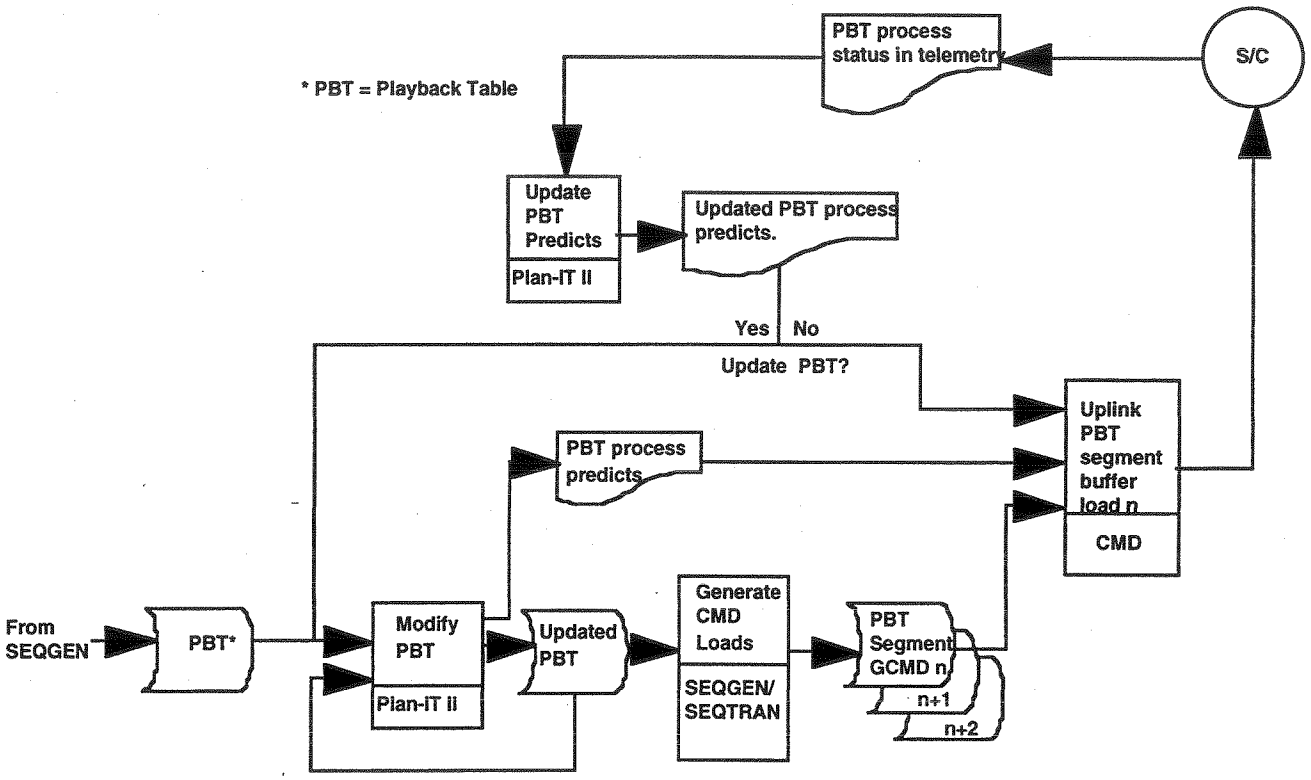


Figure 2: Playback Table Sequencing Inputs

recorder. By using the tape recorder map it produced and the table of playback directives, Plan-IT-II was able to predict the volume of data to be returned for the various onboard instruments as well as the schedule of when the data was expected to arrive on the ground. As a playback sequence progressed, actual performance, i.e., actual arrival time of data on the ground and actual data volume, was used to adjust Plan-IT-II's predicts for subsequent playbacks. If required, a new table of playback directives could be generated by Plan-IT-II for uplink to the spacecraft as illustrated in the above figure.

Cassini Team Projection Study

A short term effort (approximately two weeks) used Plan-IT-II as a forecasting tool for predicting the staffing of Cassini flight teams necessary in order to support various kinds of operations for the project during flight. Once Plan-IT-II was adapted in a couple of days, a naive user could apply it to calculate needed personnel support for various operational modes during the mission. This was a useful learning experience for our team, because we learned that if a naive user gets closely monitored tutorials to learn a subset of the Plan-IT-II commands (in this case over the course of three days), they were able to use the tool adequately.

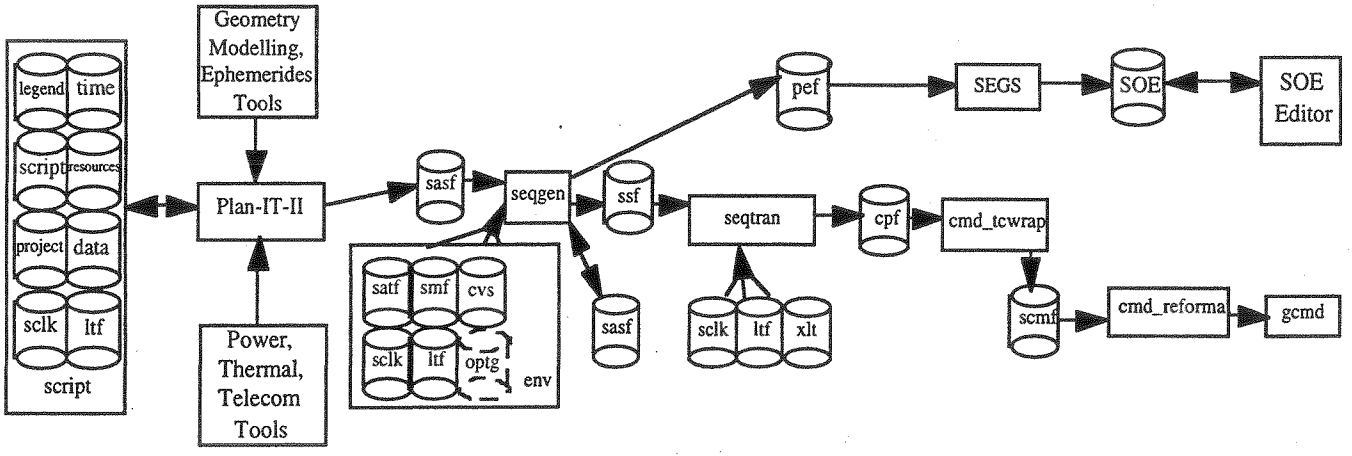


Figure 3: MPF Uplink Process Flow Through Its Tools

Mars Pathfinder (MPF)

Unlike the Galileo effort, the MPF adaptation of Plan-IT-II was used during the design of the mission as well as for operations. Plan-IT-II was used to generate many surface scenarios to determine the viability of the mission given the rover and lander designs. In fact, MPF added a sensor and almost added an additional battery as a result of running these various scenarios through Plan-IT-II. Due to MPF's relatively short development time, the project exploited Plan-IT-II's rapid adaptation capabilities. In addition, the lone MPF adapter designed and coded specialized scheduling heuristics in Plan-IT-II during this period of adaptation and scenario generation in order to ease MPF's activity sequence generation process. Both the rover and lander are modeled to the command level. Plan-IT-II's generated Spacecraft Activity Sequence file (SASF) is validated by another tool, Seqgen. However, since not all commands were represented in Plan-IT-II, those commands not often used were specified during the Seqgen run. MPF also had additional analysis tools that dealt with telecom, power, thermal, ephemerides, and rover geometry modeling which worked in conjunction with the telemetry, power, battery, and thermal models within Plan-IT-II. MPF was the first project in which external tools played a role in how some of Plan-IT-II's resource constraint models were updated.

The uplink process flow illustrates that Plan-IT-II and Seqgen shared the responsibility for generating the sequences for flight. As the level of activity on the spacecraft was low during the flight to Mars, Seqgen was used exclusively for generating the spacecraft sequences during this period. However, intensive operations on the surface of Mars were planned and generated using Plan-IT-II.

Autonomous Navigation (AutoNav) Demonstration

This was a demonstration that illustrated the potential use of auto-navigation for fire-and-forget deep space missions. Plan-IT-II's ease of adaptation made this scenario possible within a month. The scenario was a near encounter sequence with the asteroid, Melpomeme. Plan-IT-II acted as the onboard planner/sequencer that communicated with two other intelligent subsystems, an auto-navigator and an instrument observation analyzer. The Auto-Nav system was written in MatLab and the observation analyzer was a tool called Seq_Pointer. Plan-IT-II modeled a simple imaging system, tape recorder, DSN tracking passes, power, and gyros. The sequence consisted of simple activities that represented optical navigation images, engineering, slewing, playbacks of the recorded data over downlink passes, and science image observations that could either be single or mosaic images. Plan-IT-II utilized a simple heuristic whose goal was to keep the spacecraft as busy as possible utilizing a dynamic priority scheme for the various activities that it knew how to execute. However, we did not have time to incorporate any fault recovery capabilities in this demonstration. Plan-IT-II generated the sequence on-the-fly just ahead of real-time execution as the spacecraft progressed on its trajectory.

The execution loop consisted of the following: 1) Plan-IT-II would query the navigator for optical navigation image (opnav) requests for insertion into the sequence beyond the point where the sequence segment had been frozen. This request interval would vary in duration as the frozen segment did depending on where the spacecraft was in the trajectory; 2) Navigation would reply with its list of requests specifying the target ids and their coordinates in RA and DEC; 3) Plan-IT-II would communicate the series of observations that were not presently in conflict to the

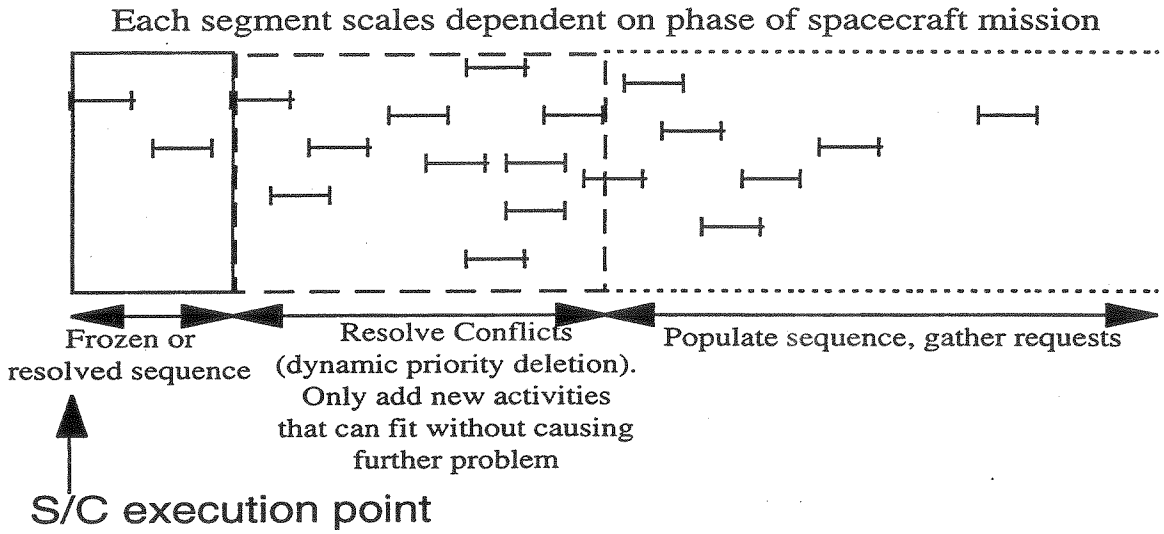


Figure 4: AutoNav Dynamic Scheduling

observation analyzer in order to determine if the image consisted of a single snapshot or required a mosaic of snapshots; 4) the observation analyzer would reply with the snapshot requirements for the observations; 5) Plan-IT-II would then adjust the activity durations for slewing and multiple snapshots if necessary; 6) if conflicts occurred, Plan-IT-II would not waste time doing a focused fix to the schedule but would do a simple priority delete; 7) when Plan-IT-II advanced the execution time bar and an activity requiring imaging was executed, Plan-IT-II would communicate with the observation analyzer to display that observation; 8) during steps 1-7, Plan-IT-II would intensively attempt to interleave playbacks (if DSN tracks were available), with engineering activities.

Each time Plan-IT-II would execute the same portion of the trajectory a different sequence of activities would result. This simple approach resulted in reasonable near encounter sequences. The communication handshake between the three major subsystems was the greatest performance impact to the system. However, we were able to generate sequences at this activity level more quickly than real-time (5 day encounter sequence would be generated and executed in 10 minutes).

Distribution and Automation Technology Advancement of the Colorado Hitchhiker and Student Experiment of Solar Radiation (Data-Chaser)

Plan-IT-II participated with the Data-Chaser experiment, which involved using several automation technologies with a three instrument payload package that operated from the Shuttle bay. The CHASER instrument gathers data from the UV, X-Ray, Lyman-Alpha wavelengths. Plan-IT-II was linked with a prototype automation tool that eventually became ASPEN (A Scheduling and Planning Environment). This complete system was called DCAPS (DATA-CHASER Automated Planner/Scheduler).

Plan-IT-II provided the graphical user interface, modelling of resource constraints, and the types of activities that would be executed to accomplish the CHASER objectives. Other constraints imposed on the CHASER instrument by the shuttle were also modelled. The ASPEN prototype was a general planning and heuristic search engine. For this experiment DCAPS would generate a first cut at the sequence of activities to be performed by CHASER. The user would make "tweaks" or manual edits to this sequence. DCAPS would then be iteratively run improving the sequence interleaved with these user edits. This was the first time a general planning and heuristic search engine tool was overlayed on top of Plan-IT-II's representation.

MicroSpacecraft Demonstration

This demonstration was built upon the AutoNav demonstration by interfacing Plan-IT-II with a prototype spacecraft sequence executive (that controlled the execution clock) and simulator. Plan-IT-II was given a simple set of commands that it would pass on to the sequence executive in order for the simulator to run. In addition, another intelligent subsystem that recognized features on asteroids and could track them was interfaced to the system. Plan-IT-II played the role of a sequence generator (as before with AutoNav), except during closest approach with the asteroid, it would hand off commanding control to AFAST and after closest approach would resume its command sequence gen-

eration process. This is a low level effort that is still ongoing.

Distributed Object Interface

Planning and scheduling of spacecraft activities often involves the iterative use of multiple software tools that deal with different kinds of domain knowledge. In order to facilitate coordinated use of these tools, Plan-IT-II and a geometrical observation planning tool were interfaced using the new Common Object Request Broker Architecture (CORBA) standard. Observations created in the observation planner were loaded into Plan-IT-II, causing the schedule and various resource constraints being modelled to change. Changes made to the observations in Plan-IT-II were immediately reflected back in the geometric display of the observation planner.

Spacecraft Infrared Telescope Facility (SIRTF)

This was another short term effort (about two weeks total) in which we demonstrated one scheduling approach that SIRTF could use in scheduling their use of a remote telescope. The problem given to us made the following assertions: 1) treat the science requests coming into the schedule as being about 300% oversubscribed; 2) rely on another tool to determine the time necessary for accomplishing the science observation (based on the instrument, the operational mode, and a set of between 5 to 30 varying attributes) [NOTE: this time included enough quiet time to account for slewing]; 3) model data storage, playback to the earth, power, and gyro maintenance; 4) split some observations into a series of cumulative observations; and 5) schedule the observations in a priority-based manner. We set up a week's worth of approximately 350 requests with two 4-hour DSN coverage periods per day. Given this set of inputs Plan-IT-II was able to generate a conflict free schedule within two minutes. Plan-IT-II's scheduling heuristic for this problem domain took about two days of effort to develop.

Deep Space One (DS1)

Due to the recent redesign of the of the DS1 mission, development time for the ground system is shorter than any other deep space mission project (less than a year). Plan-IT-II has been demonstrated over the years to quickly adapt to various problem domains. Because of the previous successes with Plan-IT-II, the DS1 ground team chose it as the tool to perform the planning and sequencing job for this mission.

Plan-IT-II is presently being adapted as the driving hub of the uplink system for the New Millennium Deep Space One (DS1) project. Plan-IT-II will be used in two phases of the DS1 mission. The first phase is the mission planning, for which Plan-IT-II will be used to generate a high level or abstract cut of the mission scenarios needed in order to accomplish the DS1 objectives. During this phase, Plan-IT-II's modeling of the spacecraft and mission constraints will actually evolve as the operational phase of the mission approaches. The resultant output of the Mission Phase will feed into the actual operations phase of the mission. During the operational phase of the mission, Plan-IT-II will be used to generate the actual sequences of commands that will be uplinked to the spacecraft. In this respect, the ground team can exploit one of Plan-IT-II's capabilities that permits the linking of actual command level relationships to already generated sequences of abstract activities at a later time of mission development. Plan-IT-II's modeling of the spacecraft as well as the constraints imposed by the mission will be extremely detailed during the operational phase of the mission in order to insure the viability of the sequences that will be sent to the spacecraft.

In the figure on the following page, Plan-IT-II will act as a driver for the whole uplink process. Plan-IT-II's job is to coordinate the incoming requests and output various products that can be fed into the gen_command part of the system. Gen_command is a batch-oriented system with the job of translating the commands from human readable form into binary form for uplinking to the spacecraft. In addition, other utilities will be used to package files properly for uploading to the spacecraft. Plan-IT-II will feed and direct both gen_command and these file uploading utilities. Plan-IT-II feeds the various sequences, and real-time commands to gen_command in the form of a series of Spacecraft Activity Sequence Files (SASF) that can be processed by Seqgen, or Spacecraft Sequence Files (SSF) to be processed by the sequence translation process, "slinc" (seqtran_2000). The reason we have this two-fold submission process is for validation testing. Seqgen is a recognized event and model simulation-tool used by spacecraft projects that has undergone rigorous testing and validation. Seqgen readily adapts for checking the commands and their argument syntax, but, requires extensive adaptation for performing the actual modeling of the spacecraft. Since Plan-IT-II has not undergone the same rigorous testing and validation, the ground team will use Seqgen to validate the syntax of the commands and their arguments. However, all of the modeling and constraint enforcement will be handled within Plan-IT-II. Once testing and validation is completed, the uplink system will be shortened by eliminating-Seqgen from the process.

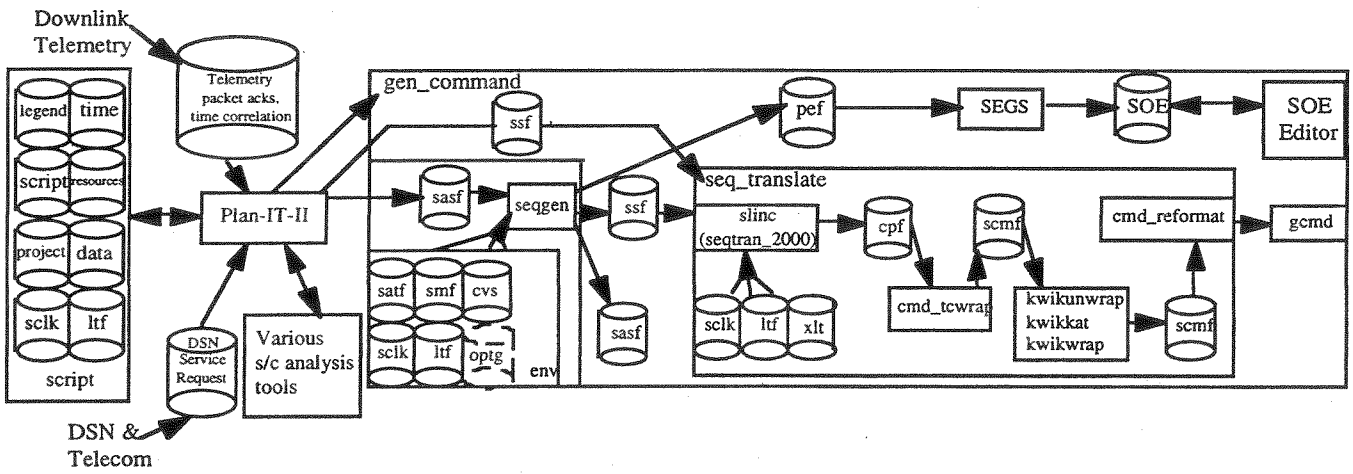


Figure 5: DS1 Sequence Process Flow through the tools

During spacecraft operations, requests come from several sources in several different forms. The downlink telemetry system produces "data received" acknowledgments and time correlation updates that must be fed into the next uplink pass to the spacecraft. Additionally, Plan-IT-II will receive from the DSN and telecom teams the viewperiod and station allocations scheduled for DS1 in the form of service requests. From the ops, engineering, and science teams Plan-IT-II integrates requests into the sequences. Plan-IT-II's integration consists of: 1) detailing the requests in the form of legal commands; 2) insuring the mix of requests doesn't violate any spacecraft models or constraints; 3) generating one or more sequences of these command in a single run; and finally 4) generating the necessary products before invoking the remaining uplink system. DS1 is the first project in which Plan-IT-II not only defines the products that are sent to the spacecraft, but in which Plan-IT-II is also responsible for insuring the viability these products.

Spin-Offs

Plan-IT-II has had the challenge of overcoming the LISP phobia of conservative, risk-averse management, so, C++ efforts were pursued to renovate one tool and build a new tool upon the Plan-IT concept. When Seqgen [14] was ported from a mainframe system that Galileo used to UNIX workstations, its GUI's conceptual origins came from Plan-IT. Another tool called Activity Plan Generator (APGEN) [15] that incorporates a subset of Plan-IT-II's capabilities was developed with a focus on making the adaptation effort and GUI more friendly to naive users. Both of these tools are in use by various projects.

Conclusions

Plan-IT owes its extensive capabilities to the years of experience and knowledge that have been incorporated as a result of being applied to a variety of tough real-world problems with extensive complex interactive modeling constraints and activity requests. Many lessons were learned and incorporated as updates and even complete revisions were made to the tool. Our main success lesson learned was to have the internal representations of the tool work in a manner that minimizes the amount of visualization required by the user in order to understand what the tool is doing when it is executing. Even though we have had success illustrating the rapid coding of specialized scheduling/sequencing heuristics for different problem domains, the following two areas need to be explored.

1. Review how the resource constraint models should rate themselves as they respond to activity requests search. Presently, we are using the same subjectively determined rating values we defined at the beginning of Plan-IT's development. This is important for the future use of Plan-IT in an onboard spacecraft application.

2. Simplify the scheduling heuristic development so naive users can accomplish it on their own. Unfortunately, Plan-IT "wizards" are still required to generate more sophisticated scheduling heuristics. If the representation within the tool could be enhanced to monitor how the user interacts with the tool in deriving a sequence, the heuristics could be learned.

ACKNOWLEDGEMENTS

All applications described in this paper were developed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. Gregg Rabideau and Dr. Steve Chien (JPL - Sec. 396) provided most of the development and adaptation work for the Data-Chaser effort. The Data-Chaser experiment was run by the Colorado Space Grant Consortium (CSGC) at the University of Colorado at Boulder.

References

[1] [Eggemeyer;Cruz 90] "Plan-IT-2: The Next Generation Planning & Scheduling Tool", Space Applications of AI and Robotics, GSFC, May, 1990. Published in Volume 7, Number3/4 1990 - ISSN 0736-5853, Telematics and Informatics.

[2] [Eggemeyer;Grenander 87] "Plan-It Applications and Knowledge Gained", Workshop on Operations Planning and Scheduling Systems for the
Space Station Era, University of Colorado-Boulder, Boulder, Colorado, August, 1987.

[3] [Eggemeyer97] "Plan-IT-2 Bible", Jet Propulsion Laboratory, Pasadena, California, Aug, 1995. Internal access only URL.

[4] [Biefeld 86] "PLAN-IT: Knowledge-Based Mission Sequencing," E. Biefeld, Proceedings, *Advances in Intelligent Robotics Systems Conference*, Cambridge, Mass., October, 1986.

[5] [Eggemeyer;Grenander 87] "Plan-It Applications and Knowledge Gained",*Workshop on Operations Planning and Scheduling Systems for the Space Station Era*, University of Colorado-Boulder, Boulder, Colorado, August, 1987.

[6] [Zamani; George;Collins; Zimmermann 89], "Spacecraft Activity Planning Tool Using Object Oriented Techniques", Proceedings, Tools '89, Paris, France Nov. 13-15, 1989.

[7] [Bahrami;Biefeld;Costello;Clein 86] "Space Power System Scheduling using and Expert System", Proceedings, *21st Intersociety Energy Conversion Engineering Conference*, San Diego, CA, August, 1986.

[8] [Eggemeyer;Bowling 87] "Deep Space Network Resource Scheduling Approach and Application," Space Applications of AI and Robotics, GSFC, May, 1987.

[9] [Dias;Henricks;Wong 87] "Plan-It: Scheduling Assistant for Solar System Exploration", *Telematics and Informatics*, Vol. 4, No. 4, pp. 275-287, 1987.

[10] [Katz;Brooks 87] "Understanding Natural Language for Spacecraft Sequencing", *Spaceflight, the Journal of the British Interplanetary Society*, Nov., 1987.

[11) [Katz 87] "START Natural Language System", MIT AI Lab Memo, 1987.

[12] [Hull;Peters;Davis 93], "EOS Distributed Planning and Scheduling Prototype", Proceedings of the AIAA Computing in Aerospace 9 Conference, San Diego, CA.

[13] University of Colorado at Boulder, Jet Propulsion Laboratory, Goddard Space Flight Center, Earth Observing System Distributed Planning and Scheduling Prototype Lessons Learned, EOS Library, NASA/Goddard Space Flight Center, Greenbelt, MD, June 1993.

Resource Allocation Using Fine-Grained Demand Models

Kutluhan Erol and Robert Kohout

Intelligent Automation Inc.
2 Research Place #202
Rockville MD, 20850
kutluhan@i-a-i.com, kohout@i-a-i.com

Abstract

Multi-agent problem solving using market mechanisms is a primary focus area of Intelligent Automation Inc. We have been developing agent-based planning and scheduling techniques, and investigating their feasibility in manufacturing and transportation domains. These domains involve online streams of jobs being executed in dynamic and uncertain environments. Many NASA applications, such as scheduling DSN antenna facilities, and mission planning share these characteristics. We believe that market mechanisms greatly facilitate problem solving under these conditions, insofar as they provide a means of localizing decision making. This paper describes how we use fine-grained models of cost to facilitate computational resource allocation in Cybele, an agent infrastructure that we have been developing to support multi-agent processing on a distributed network of computers. This technique of resource allocation using fine-grained demand models may be of utility in finding high quality solutions to the DSN scheduling problem.

The NASA Demand-Based Autonomous Network Scheduling System

(Chien,et.al, 1997b) describes the DANS rescheduling and resource allocation system, which is designed to support incremental changes in a complex schedule of satellite-tracking antennas and their supporting subsystems. The Deep Space Network (DSN) antennas that it schedules are critical resources, so the DANS system decomposes the problem into a two-level hierarchy, where the scheduling of supporting subsystems is performed in the context of a particular solution to the antenna-scheduling problem. Assignments of antennas to perform specific tasks in fixed time windows are referred to as *tracks*. Given a new activity to schedule, DANS exhaustively searches through the set of constraint-satisfying antenna schedules, which corresponds to a set of tracks, and ranks them as a function of the disruption they cause to the current schedule. Tasks that can be inserted to the current schedule without requiring the cancellation of any existing tracks are treated as if they have zero cost. Where there are conflicts, existing tracks may be preempted based upon a priority-level analysis. More important task are given low priority numbers, and tasks may only preempt tracks of tasks with higher priority numbers. These priority levels appear to be small integers, in the range of roughly 0-9. When conflicts do occur, solution costs are ranked according to the formula

$$\text{Cost} = (\mathbf{NAD} * \text{priority}) / (0.9\,\mathbf{NAD} + 0.1)$$

Where **NAD** is the number of deletions required to schedule the current activity. When tracks are deleted, the scheduler tries to immediately reinsert them in to the schedule. In addition to this task-at-a-time top-down scheduling, DANS can also be asked to make bottom-up repairs to its schedule in an effort to recover from unanticipated events, such as equipment failure or weather conditions.

We believe that the cost model adopted by DANS can be improved significantly. The priority scheme described in (Chien,et.al, 1997b), allows only very coarse comparisons between solutions. In our work on scheduling systems, we have adopted that view that tasks should be assigned fine-grained values, and the scheduler should be designed in such a way that this information can be used to produce high quality solutions. This approach has several advantages. Determining the demand a job has for resources is more natural and direct than finding the appropriate priority level needed to produce the desired results. It requires no specialized knowledge of the way in which the scheduler interprets priority levels. .In many cases, it amounts to asking a customer how much he is willing to pay. This approach enables us to employ a number of well-known market mechanisms in the search for near-optimal solutions. It allows us to more accurately account for the relative worth of the various tasks, and enables us to apply well-known principles of market economics to scheduling problems.

In this paper, we exploit the advantages of this cost model to develop a set of algorithms for dynamically balancing the computational loads on a set of loosely-coupled processors hosting a community of cooperative software agents. In the special case where there is only one

computer, this problem is fairly similar to one which must be addressed by every modern operating system. Conventional solutions to this problem almost always use the sort of coarse-grained scheduling that DANS employs in its top-down scheduler. As a result, it can be very difficult to optimize CPU usage, even when the relative demands of the various processes in the system are known precisely. We examine the case of autonomous agents because it is both fundamental to the work we do at IAI, and because it allows us to consider cases where the migration of tasks between processors is a valid and realistic consideration. While the problem of optimizing CPU usage is not isomorphic to the problem of scheduling satellite arrays, there are a number of similarities. In both cases, there are large numbers of tasks, or consumers, that compete for scarce resources, and the problem is one of finding high levels of global utility through some assignment of resources to consumers. In both cases, some tasks are deemed more valuable than others. Granted, there are fundamental differences in the two problems, and we are not claiming that our results are directly applicable to DNS scheduling. However, we do believe that, for purposes of achieving high-quality assignments of resources to consumers, the fine-grained model of cost and demand has tremendous advantages over coarse-grained systems. In this paper, we show this in the context of managing CPU allocation.

Computational Resource Allocation and Dynamic Load Balancing

Agents are persistent and goal-oriented, thus they are sensitive to the availability of computational resources. However, the benefit an agent receives from additional CPU cycles varies widely, depending on the importance of the tasks he is working on, the due dates, and also how long he has been working on them (law of marginal returns). Thus the demand for CPU cycles can be different at each agent, and it varies over time. Similarly, the amount of computational resources available on the network can fluctuate as the computers go down/up, or additional workload is added to them. Allocating resources to agents in an optimal way becomes a very intricate problem under these considerations.

Load-balancing and CPU optimization can be considered at three different levels:

- In a single agent community/computer
- Among agent communities on a local area network
- Among local-area networks of agents connected via the Internet.

Within a single community, agents have to time-share the CPU. The problem to decide is what percentage of the CPU cycles each individual agent should get, in order to make the most profitable use of the available CPU cycles. This clearly goes beyond the rudimentary priority-based round-robin scheduling employed by most operating systems.
In a local-area network of agent communities, the problem becomes deciding how to distribute/migrate agents among agent communities in such a way that the aggregate computational capacity on the network is optimally utilized. The optimal solution is at an equilibrium point, where the return for an additional CPU cycle(marginal utility) is the same at each agent community. It is desirable to maintain the equilibrium point with minimal overhead, while being responsive to fluctuations in the capacity of computational resources, as well as the fluctuations in the demands of individual agents. It is also important to take into consideration the cost of migrating agents.

In this paper, we do not consider the third level, which is among local-area networks, connected via the Internet. In the abstract, this is the same problem as load-balancing in a local-area network, with higher costs of communication and agent migration. Thus the same techniques apply. Note that the clustering of load-balancing into several levels makes our approach scaleable to very large systems.

In the next two sections, we will present the technical details of CPU allocation within an agent community, and then the load-balancing protocols across agent communities on a local area network.

CPU Allocation within an Agent Community

Resource allocation has been a central problem addressed by microeconomic theories. We draw heavily from that body of work. While economists have focused on existence of equilibrium points, our focus is on the computational expedience of finding and maintaining the equilibrium under changing conditions. The time scale makes a big difference: assigning CPU cycles optimally requires responsiveness in the order of seconds, as opposed to in the order of days, or even months in commodity markets. Thus any practical solution must be extremely fast to compute. On the other hand, agents are engineered to have very simple structure compared to people. Hence behavioral assumptions, such as rationality, apply better to agents than to people. As a direct consequence, we expect economic market models to be even more applicable to agents than to people.

In an agent-based application, where each individual agent is a profit maximizing entity, each agent operates by selling his services. In order to provide his services, an agent will need to buy services from other agents. For example, in a factory setting, a particular agent may have the capability to produce a part, but in order to perform this operation he will need to buy input materials as well as time on the proper machine, tool, and fixture, who can perform the operation. In such an environment, it is natural to treat

computational resources, just as any other service an agent needs to buy. In making such decisions, the agent needs to decide whether to buy from an expensive, faster machine, to buy now versus at a later time, or choose not to buy at all, if the prospective job is not profitable. The same applies to buying CPU cycles. The agent must decide how much and when to buy/release CPU cycles to maximize his profit.

Let Q denote the computational capacity (cycles/sec) available to an agent community. Assume a mother agent of the community that will be responsible for monitoring the change in available capacity and allocating it among her children agents. Assume the current number of agents in a community to be n.

Each agent maintains a demand function, $d_i(q)$, which denotes the rate (in dollars) he is willing to pay for one additional CPU cycle/sec, at his current consumption rate of q cycles/sec. Thus each agent must know how much CPU cycles are worth to him. This is a strong assumption for traditional software; however, agents, which are designed to operate in a market model for profit maximization, would be aware of their computational needs.

A market equilibrium point $<p,q_1,...q_n>$ is defined with the following equations:

$$\Sigma(q_1,...q_n) = Q$$
$$q_i >= 0 \qquad for \quad i= 1..n$$

$$d_i(q_i) = p \text{ or } [q_i = 0 \text{ and } d_i(0) < p \,] \qquad for \quad i= 1..n$$

where p is the equilibrium market price. In other words, the CPU cycles are sold to the agents at p dollars per unit. At the equilibrium point, all the available capacity is distributed among the agents. No agent can have a negative share. There is an equilibrium price p, such that every agent with a positive share values his next unit CPU cycle at the same price p. Thus there is no incentive to swap CPU cycles. The agents whose share is 0 are not willing to buy any cycles at the current market price p.

Theorem 1. When the demand functions for individual agents are continuous, and strictly decreasing, there exists a unique equilibrium point.

Synopsis of Proof. The fact that the demand functions are continuous and strictly decreasing implies that the demand functions are one-to-one and their inverse functions exist. The inverse functions are also continuous, and strictly decreasing. Let $\underline{d}_i(p)$ be the inverse function for $d_i(q_i)$. The function $\Sigma[\underline{d}_i(p)]$ will also be strictly decreasing and continuous, and thus there exists a unique equilibrium point where $\Sigma[\underline{d}_i(p)] = Q$.

An interesting property of the market equilibrium point, is that it is also a Nash-equilibrium point. That is, no agent will benefit from lying about his demand function, when the remaining agents are honest about their demand functions, as stated in the following theorem.

Theorem 2. The market equilibrium point is also a Nash-Equilibrium point.

Synopsis of Proof. Note that the total amount an agent pays for his CPU cycles is only affected by the quantity he is willing to buy at the market equilibrium price. If the agent has declared a higher demand at that point, then he will be forced to buy additional CPU cycles for a higher price than he is willing to pay. If he has declared a lower demand, then, at the market price, he will miss the chance to buy CPU cycles at a lower rate than he is willing to pay. He loses in either case.

We have presented the equations that determine the market equilibrium point, and proven that there exists a unique solution when the demand functions are constrained to be continuous and strictly decreasing. However, in order for our approach to be practical, we must be able to effectively compute the equilibrium point in the order of milliseconds, in response to fluctuations in the demands from agents, as well as the fluctuation in the available computational capacity.

Obviously, arbitrary demand functions are hard to represent in data structures, and do not lend themselves to closed-form solutions that are efficient to compute. We identify a class of demand functions parameterized with few variables that lead to closed-form solutions which are efficiently computable. This class of functions is also rich enough to represent a wide range of demand functions, and has the following general form:

$$d(q) = a/(b+q); \quad q>=0$$

Depending upon the values the parameters a and b take, this function can assume many different forms. If b is very large, it will appear as a flat line in the operational range. If b is very small, it will appear almost like a vertical line, and in between, it will have a hyperbolic form. The absolute value of the demand can be adjusted with parameter a. Furthermore, this leads to a very compact function representation: each agent needs to convey only the values for a and b. Note that this function is continuous and strictly decreasing, thus there exists a unique equilibrium solution, which tells us the optimum way to allocate the available computational capacity.

This form also leads to a closed-form solution that is easy to compute:

$$q_i = a/p - b_i$$
$$\Sigma[q_i] = Q$$
$$\Sigma[a_i]/p - \Sigma[b_i] = Q.$$

Let $\Sigma[a_i] = A$ and $\Sigma[b_i] = B$.

$$A/p - B = Q$$
$$p = A/(B+Q)$$

This particular formula does not give the exact equilibrium price: it ignores the constraint that $q >= 0$. A careful examination of the solution reveals that the agents with low demand functions will attempt to operate with negative q_i and sell CPU cycles. If we can tell which agents will have non-zero capacity shares at the equilibrium price, and include only them in the formula, we will have the exact equilibrium price. Fortunately, there is an efficient way to compute the actual equilibrium point, as outlined in the algorithm below:

Find Community Equilibrium
1. Sort agents in decreasing order of a/b.
2. Let A = 0, B = 0; k = 1; P = 0;
3. while $(P < a_k/b_k)$ and $(k <= n)$
$\{A = A + a_k;$
$B = B + b_k;$
$P = A/(B+Q);$
$k = k+1\}$
4.for j = 1 to k-1 do
$q_j = a_j/P - b_j;$
5. for j = k to n do
$q_i = 0;$

The basic intuition of this algorithm, which computes the exact market equilibrium point is as follows: Initially the set of agents with non-zero capacity allotment is empty. Iteratively add the agent who is willing to pay the highest price for his first CPU cycle to the set, as long as that price (computed by formula a_i/b_i) is above the current market equilibrium price. The iteration terminates, when none of the remaining agents are willing to buy any CPU cycles at the current equilibrium price.

The sorting part of the algorithm runs in $O(n \log n)$ time; however, it is needed only once. From then on, updates will take linear time. The iteration in the algorithm is also linear.

Updates occur in response to following events:
- as new agents are added to the community
- as agents in the community are terminated
- as agents change their demand functions
- as the computational capacity available to the community changes.

CPU Allocation and Load-balancing Across Agent Communities

In this section, we focus on how to best utilize the computational capacity distributed in a local area network. Each agent community in the network has some capacity, and a number of agents. Even though each community is optimizing its CPU cycles using the technique detailed in the previous section, this may not correspond to the global optimum. In a community where demand is high and the capacity is low, the equilibrium price will be very high. In contrast, there may be other communities where the demand is relatively low, and the capacity is large. In such communities the equilibrium price would be low, and the agents in the high-price community can make better use of some of the CPU cycles at the low-price community. Thus we would like to be able to migrate agents from high-price communities to lower priced communities, until all communities have the same equilibrium price. This problem is complicated by the fact that the demand and capacity at each community will be continually changing, due to both external factors (behavior of agents, computers going down, etc.), and internal factors (agents migrating across communities). Agent migration itself has a significant duration, and requires CPU cycles. There is also a tradeoff between the responsiveness of the system, and its stability.

We will first describe the equations that define the theoretical global optimum. This will assume:

- All the information is available at a central location
- There is no latency in the data
- Agent migration is instantaneous

Naturally, these assumptions are not valid in a distributed network environment. We will present techniques that do distributed load-balancing, without these assumptions. When tuned, these techniques will keep the system close to the global optimum, without putting it into oscillation.

One can view a network of agent communities as a single, virtual agent community that houses all the agents in the system, whose computational capacity is the sum of the capacities of individual communities. With this translation into a single virtual community, the allocation techniques developed for a single community can be applied directly to compute the equilibrium price, and the individual share of CPU cycles that each agent will receive. At that point, we have a partitioning problem in our hands: distribute the agents to communities in such a way that the difference between the capacity of a community, and the aggregate capacity share of the agents assigned to that community is minimized. Once the agents are distributed to communities, the equilibrium price at each community, and the share of each agent can be computed. Naturally, there will be some difference in the price level of communities, due to the fragmentation when each agent is assigned to a community. Thus the theoretical equilibrium may not be attainable.

Next, we present a protocol that performs dynamic load balancing. This protocol can handle several important issues:

- New agent communities can appear anywhere in the network
- Existing communities may disappear as their host computers goes down or otherwise become unavailable
- The computational capacity available to a community changes
- New agents are created, terminated
- Existing agents' demand for CPU cycles change

Furthermore, this protocol is parameterized so that it can be tuned to the frequency of changes in the system. In future research, we will address how to dynamically adjust those parameters to changing patterns in the system.

The first step towards moving the system towards the global optimum (equilibrium) point involves estimating the equilibrium price. The theoretical analysis outlined previously requires the information about the capacity of each community and the demand function for each agent in the system. It is not practical to maintain this large body of information at a central location, because it would be severely out of date most of the time. Instead we have devised methods of abstraction to compress this information so that it can be shared among agent communities.

Recall that the market equilibrium algorithm within a single agent community computes two factors: $A = \Sigma[a_i]$; $B = \Sigma[b_i]$. A and B are respectively sums of parameters a_i and b_i for the agents in that community with non-zero capacity share at equilibrium point. Thus A and B contain the summary information on the demand functions of the individual agents in that community.

Let A_i be the sum of a_i of the agents in agent community i, and let B_i be the sum of b_i parameters of agents on community i. Let Q_i be the computational capacity available to the community i. The estimated equilibrium price P is given by the formula:

$$P = \Sigma[A_i])/(\Sigma[B_i] + \Sigma[Q_i] \qquad \textit{(equilibrium price formula)}$$

This formula is only an estimate, and not the actual equilibrium price, because it presumes that at the global equilibrium point the set of agents with non-zero share allotment will be the same as in the initial state. Recall that the closed-form solution for the single community equilibrium was inexact, and this was corrected by the Find-Community-Equilibrium Algorithm. We cannot utilize the same technique in the multiple community version, as it would require that the demand functions for each agent be known at a centralized location. However, in most cases, the closed-form solution provides a close approximation, which improves as the system moves towards optimality, and it is exact when the system is operating at market equilibrium. Thus it can be successfully used as a guide for converging to the equilibrium point.

Our load-balancing protocol requires that each mother agent be able to estimate the global equilibrium price. Thus each mother agent will maintain a list containing $<A_i,B_i,Q_i>$ for all communities in the system. We will call this list as the *system load table*. Each mother agent periodically, and in response to significant changes, multicasts her new $<A_i,B_i,Q_i>$ values to the other mother agents, so that this information remains reasonably up to date and accurate. Mother agents also monitor the termination and creation of other agent communities with the same set of messages.

Using the data in the system load table, each mother estimates the global equilibrium price using the above equilibrium price formula. Each mother agent also estimates he equilibrium price at each other agent community using the formula $p_i = A_i/(B_i + Q_i)$. This formula gives accurate answers as long as the parameters are up-to-date. In actuality, it will be a close approximation, because minor fluctuations in those parameters are not reported, in order to minimize message traffic.

Our load-balancing protocol involves two parameters *overload factor* and *underload factor*:

> A community is overloaded if $P_i >$ *overload factor* $* P$, and underloaded if $P_i <$ *underload factor* $* P$.

Where P is the estimated global equilibrium price.

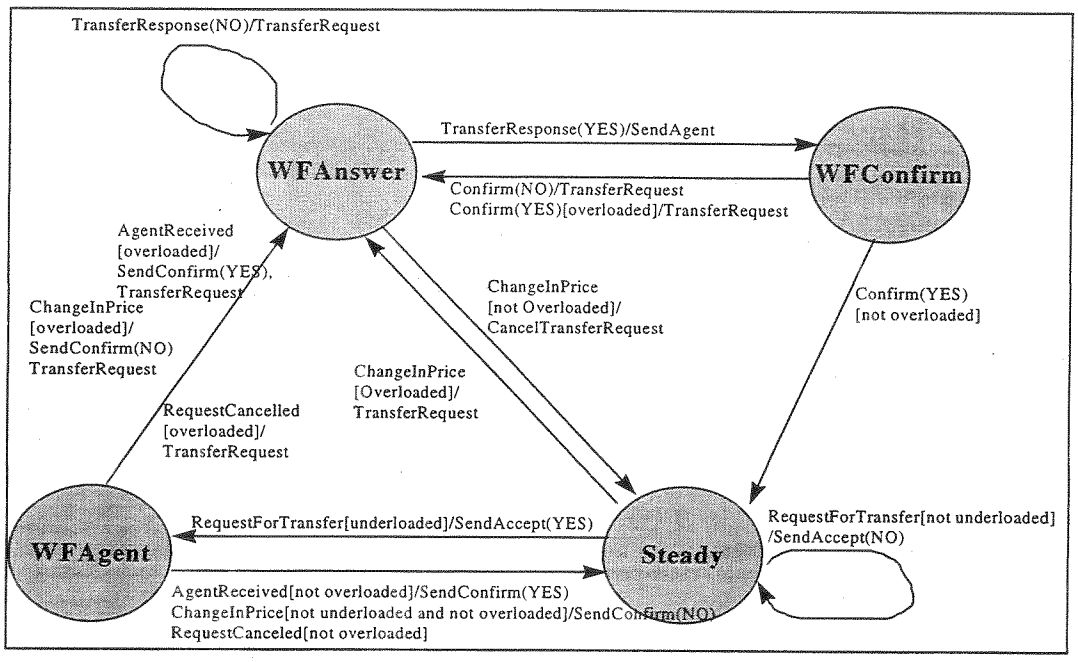


Figure 1. State-Transition Graph for Load-balancing Protocol

Overloaded communities attempt to transfer some of their agents to other communities, and a community will accept incoming agents only if the community is underloaded. For values of load factors close to 1, the system attempts to transfer agents eagerly in order to reach close to the optimum point. This is suitable for fairly stable systems with low frequency of fluctuations. For systems with high frequency of fluctuations, it is more appropriate to set the load factors so that the system behaves more conservatively in migrating agents to keep close to the global optimum.

The dialog among mother agents for load balancing proceeds as follows: A mother agent who finds she is overloaded (due to increase in the demand functions of her agents, or a decrease in her computational capacity) probabilistically selects an underloaded mother. The more underloaded a mother is, the likelier she will be selected. The overloaded mother sends a transfer request to the selected mother .If the transfer request is rejected, and the mother is still overloaded, she will probabilistically select another mother. If the transfer request is accepted, she will send one of her agents across and wait for confirmation that the agent is successfully migrated to the other side.

Figure 1 presents the state transition diagram for a mother agent's load-balancing activity. There are a number of short-cuts in the dialog. In particular, mothers can change their minds about sending an agent, or accepting an agent, depending on changes in the market prices.

There are four states:

1. **Steady**: This is the state where the mother agent remains as long as she is not overloaded, and she is not involved in an agent transfer
2. **WFAgent**: This is the state where a mother agent has accepted a new agent and waiting for his arrival.
3. **WFAnswer**: This is the state where the mother agent has asked another mother to take an agent from her, and she is waiting for the response.
4. **WFConfirm**: This is the state where the mother agent has sent one of her agents to another mother, and she is waiting for confirmation that the agent was successfully revived on the other side.

There are also time-out transitions, not displayed in the diagram, so that the protocol can robustly operate in the presence of lost messages, or agent migration failures.

Conclusion

In cases where the relative demand of processes is known, there is no simple way to reflect this evaluation in the coarse-grained priority, round-robin schedulers found in almost all modern time-sharing operating systems. In this paper, we have shown that it is possible to optimally schedule single processors when consumer priorities are allowed to accurately reflect demand. We have also described a way to find near-optimal assignments of jobs across multiple processors, when jobs are allowed to move between processors. In our opinion, the advantages of fined-grained cost models are indisputable in this sort of complex optimization problems. As described in (Chien, et.al, 1997), DANS falls into this category, but for unstated reasons, the optimization issues are deemed secondary. In

many cases, the deficiencies of this approach can be addressed via an intelligent analysis of the domain, and a careful assignment of priorities. However, this can be an arduous task, and it can be simpler to work with a straightforward demand model.

There are other factors that the DANS scheduler must take in to account. For example, schedule stability is highly desirable, and the current system maximizes stability for the highest priority jobs. Market models suggest an alternative approach, in which contract penalties are determined on a task-by-task basis, and encourage stability while still allowing the scheduler some flexibility. These penalties can be adjusted to reflect the desired level of stability.

The top-down approach adopted by DANS is already very time-consuming. When new tasks are being considered, DANS enumerates all of the constraint-satisfying possibilities, and then evaluates the impact of each of these on the current schedule. A finer-grained cost model in this context would greatly complicate the problem, to the extent that the approach would likely be no longer feasible. In our opinion, this is a feature of the top-down search strategy that DANS employs. A number of authors, e.g. (Fisher and Muller, 1995, Sandholm, 1993, and Wellman, 1992) have demonstrated that contract net based, distributed problem solving techniques can be used to achieve high levels of global performance in complex optimization problems. Our work at IAI (see, for example, Baker,et.al, 1997), is focused almost exclusively in this direction, which can be viewed as bottom-up scheduling and optimization. One advantage of this approach is that it is well-suited to fine-grained models of both cost and demand. In this paper, we have tried to show the utility of such models in a particular context, and suggest that they be considered in the context of satellite array scheduling.

Acknowledgments

The work described in this paper has been performed by the multiagent research group of Intelligent Automation Inc. Our teaming partners University of Cincinnati, Industrial Technologies Institute, and Flavors Technologies have tremendously contributed to our manufacturing effort. This work has been in part funded by NASA SBIR NAS8-97035, DARPA Agile Manufacturing Program BAA 94-31, Department of Commerce SBIR contract 50-DKNB-6-90112. The views expressed in this paper are solely those of the authors, and do not necessarily reflect the views of the sponsor agencies.

References

Baker, B., Parunak, V., Erol, K. 1997. Manufacturing over the Internet and into Your Living Room: Perspectives from the AARIA Project. Forthcoming.

Chien, S., Hill, R. W. Jr., Govindjee A., Wang X., Estlin, T., Griesel, M.A., Lam, R., and Fayyad, K.V.1997a. A Hierarchical Architecture for Resource Allocation, Plan Execution, and Revision for Operation of a Network of Communications Antennas. *Proc. 1997 IEEE Int'l Conf. on Robotics and Automation.*

Chien, S., Lam, R., and Quoc V,, 1997b. Resource Scheduling for a Network of Communication Antennas *Proc. Of the IEEE Aerospace Conference,* Aspen, Co.

Fischer, K.,and Müller, H. J. 1995. Cooperative Problem Solving in the Transportation Domain. in Ulrich Derigs (ed.) *Proceedings of the International Conference on Operations Research.*

Sandholm, T. 1993. An Implementation of the Contract Net Protocol Based on Marginal Cost Calculations. *Proceedings of the Eleventh National Conference on Artificial Intelligence.*

Wellman, M. 1992. A General-Equilibrium Approach to Distributed Transportation Planning. *Proceedings of the Tenth National Conference on Artificial Intelligence.*

AI Planning for Just-In-Time Training in Space: ESA's Integrated Learning System

T. J. Grant, M. Verhoef, L. P. Gale

Origin Netherlands b.v.
P. O. Box 1444
NL-3430 BK Nieuwegein
The Netherlands
{Tim.Grant; Marcel.Verhoef; Leslie.Gale} @nl.origin-it.com
http://www.origin-it.com

Abstract

Advanced planning and scheduling technologies have been deployed largely in operational space applications to date. Taking the European Space Agency's Integrated Learning System[1] as an example, this paper demonstrates that operational support functions - most notably training of crew and ground support personnel - can also benefit from the application of such technologies. The Integrated Learning System combines AI planning and Web pages to deliver personalized training at geographically-distributed locations. AI planning is an enabling technology for Just-In-Time Training, and delivering Web pages over the Internet/Intranet is an enabling technology for On-the-Job Training. Designed to be domain- and platform-independent, the system has been implemented and applied to a full-size domain in support of the European Space Software Development Environment. By replacing the existing training material, the system could be readily applied to other domains. We propose training of the International Space Station crew and ground support personnel as an obvious future application.

Introduction

Advanced planning and scheduling technologies have been deployed to date largely in operational space applications, as shown by this workshop's Call for Participation. By contrast, this paper demonstrates that operational support functions, such as training, can also benefit from the application of advanced planning and scheduling technologies. In particular, we show that AI planning is an enabling technology for delivering Just-In-Time-Training (JITT) in-orbit and on-ground. The Integrated Learning System (ILS), implemented for the European Space Agency (ESA), is used as an illustrative example.

The ILS delivers training material at geographically-distributed locations via the Internet or Intranets. AI planning technology is employed to generate personalized training courses in real-time, making JITT feasible. When the training material is in the form of Web pages, the training courses can be delivered over the Internet or an Intranet for execution at the trainee's normal workplace using a standard Web browser, thus enabling On-the-Job Training (OJT).

Designed to be domain- and platform-independent, the ILS has been implemented fully. It has been applied to a full-size domain in support of the European Space Software Development Environment (ESSDE), namely the development of on-board software in Ada using ESA's Hierarchical Object-Oriented Design (HOOD) method and the LOTOS and RAISE formal methods. The ILS implementation, complete with this ESSDE training material, was delivered to ESA and accepted in March 1997.

The ILS could be readily applied to other domains by replacing the ESSDE training material with training material for another domain. In this paper we propose training of crew and ground support personnel training for the International Space Station as an obvious future application of the ILS.

This paper focuses on the AI planning aspects of the ILS. There are six chapters. Chapter 2 outlines the ESSDE. Chapter 3 describes the ILS, including its software architecture and the representation of training knowledge as learning units and instruction sequences. Chapter 4 discusses the AI planning issues arising during ILS development. Chapter 5 identifies other future applications, with an emphasis on ISS crew and ground support personnel training. Chapter 6 draws conclusions.

European Space Software Development Environment

Ada is ESA's preferred programming language for on-board software for the European element of the International Space Station: the Columbus Orbital Facility (COF). The Columbus Software Development Standards

[1] The views expressed in this paper are those of the authors and do not necessarily represent those of ESA.

(ESA, 1993) stipulate that SADT[2], HOOD, Petri Net, and Extended Entity-Relationship methods must be used in on-board software design.

Stating a preference for Ada and HOOD is in itself not enough to ensure that Ada will be used by on-board software developers. They also require a matching software development environment. Recognizing this, ESA instituted the European Space Software Development Environment (ESSDE) programme with two projects:

- An Ada/SADT/HOOD tool was developed in the ESSDE Reference Facility Project, and
- Ada training was a key theme of the ESSDE Advanced Methods and Tools (AMT) project, together with an investigation of the incorporation of formal methods such as LOTOS (Language of Temporal Ordering Specification) and RAISE (Rigorous Approach to Industrial Software Engineering).

The ILS has its roots in the Ada training programme defined in the ESSDE AMT project. The objective of the Ada training programme was to create the necessary conditions for optimal use of Ada. A survey of Ada users disclosed the following obstacles to optimal use of the language (Gale, 1993):

- The number of users familiar with Ada, HOOD and formal methods is vanishingly small.
- Users were very often unfamiliar with the full range of constructs available in Ada.
- Familiarity with other high-level languages is an interference.
- There are few methodologies for transferring from design notations to Ada constructs.
- Training of users in Ada is often inadequately planned and applied ad-hoc.
- Continuing support is rarely available when required.

The solution to these obstacles was seen as the integration of a training system (now the ILS) into the software development environment, capable of providing (Gale and Mol, 1995):

- individualised and group training.
- trainee-, task-, and role-oriented training, i.e., responsive respectively to the trainee's skills, to the skills needed to perform the current task, and to the level at which the skills are to be performed.
- feedback and self-assessment.
- structured assistance in the management of training.

Integrated Learning System

Software architecture

The ILS has been implemented according to ESA's Software Engineering standards (ESA, 1991) using object-oriented design methods. Commercial Off-The-Shelf (COTS) development tools have been used, resulting in a low-cost, portable and extendible Computer-Based Training (CBT) solution. Runtime versions are available for SunOS 4.1.3, Solaris 2.5, and Windows NT/95. As delivered to ESA in March 1997, the ILS contained some 340 learning units covering software engineering in Ada, HOOD and LOTOS.

The ILS has become a generic CBT tool. Training can be provided on-the-job and just-in-time. Training sessions can be tailored to specific trainee needs depending on the trainee's existing skills and on the task he/she faces. Standardized training sessions can also be delivered to many trainees, either simultaneously or at times that suit the individual trainees.

The ILS is so named because it can be integrated into a larger system. In the ESSDE application, the ILS would be integrated into the ESSDE toolset. Integration is made possible by the ILS capturing data during the execution of training. The data includes time spent in each part of a training course, the number of failures made in each test, and the trainee's inputs. This data could be used to score the trainee's performance in known training courses, enabling successful trainees to be certified for mission-critical operations. Additionally, the data could be used on-line to bring in instructor assistance when a trainee needed it, or off-line to provide feedback to curriculum and courseware designers. Uniquely, the data enables an AI planning algorithm to modify the training course in real-time in response to the trainee's needs.

An ILS installation consists of the following elements:

- *Browsing and Tutoring Client.* Each trainee would have a COTS Web browser such as Netscape or Internet Explorer.
- *Browser and Tutoring Server.* A COTS Web server makes the training material available on the WWW or an Intranet to which the trainees have access.
- *Learning Unit Data Base (LUDB).* The LUDB contains the training material, represented as a collection of LUs. The training domain of the ILS installation is changed simply by changing the contents of the LUDB.
- *Authoring Support Environment (ASE).* The ASE is used to create the LUs, if necessary by conversion from existing documentation. A COTS HTML editor may be used.
- *Knowledge Data Base.* The knowledge data base contains the training courses, represented as instruction sequences. The curriculum can be changed without changing the training material in the LUDB by adding or removing courses in the Knowledge Data Base.
- *Planning and Evaluation Environment (PEE).* The PEE provides the facilities for defining instruction sequences and for evaluating the results of training sessions. It has been implemented in C with TCL/TK for the GUI and runs on X-Windows under SunOS and Solaris. At present, the PEE supports manual construction of training courses. Automated construction using an AI planning algorithm has been

[2] Also known as IDEF0.

prototyped, but - because of funding restrictions - has not yet been integrated into the PEE.

Selected ILS-related project documents are available on the European Space Technology and Research Centre (ESTEC) Web site at[3]:

http://www.estec.esa.nl/wmwww/WME/CBTraining/atp/index.html

ILS functionality

The ILS provides functionality to support two types of end-user:

- *Courseware designer.* The courseware designer would create and edit training knowledge using the ASE, and define curricula and individual courses using the PEE. In addition, the courseware designer would evaluate the results of training sessions also using the PEE.
- *Trainee.* Trainees would follow guided training courses using their Web browser and the ILS's Knowledge Data Base and LUDB. For refresher training, they would perform structured browsing using their Web browser and the LUDB. Given the appropriate permissions, they could also perform unstructured browsing of the training material outside the ILS.

Additional end-user roles could be identified. For example, the instructor role might be distinguished from that of the courseware designer, e.g. by making test result evaluation an instructor task. Considerations of practicality will also lead to the identification of maintenance roles, such as data base administrator.

The ILS retains the useful functionality of Web-based training systems:

- Training material can be browsed at a distance from where it is stored.
- Conceptual links between items of training material can be implemented statically as hyperlinks.
- Externally-provided material can be incorporated in training courses.
- Access to the training material can be restricted using Intranet or Extranet techniques.
- Search engines can provide users with the facility to search large quantities of training material.

The advantages of the ILS over conventional Web-based training systems are as follows:

- The training material is structured by means of learning units, the role-task model, and intellectual skill and training base classes. This indexing structure can be used to guide browsing.
- Training material can be sequenced into training courses for guided training.
- A given item of training material (i.e. a learning unit) can be part of multiple training courses, minimising duplication and inconsistencies between training courses.
- The trainee's understanding of the training material can be tested, with tests being embedded into training courses.
- Test results and other data (e.g. dwell times in individual learning units) can be captured and made available for other systems. For example, the test results of individual trainees can be used for certification, and captured data from multiple trainees can be used to evaluate the effectiveness of the training material and courses.

Possible extra functionalities that could be implemented in the ILS include:

- Sending test results by email to an instructor, enabling the instructor to assist trainees that are having difficulties with a particular part of a training course.
- Synchronising the execution of a training course for trainee and instructor, enabling them to work on the course together at a distance.
- Coupling the execution of training courses with simulations, emulations, and operational applications.
- Automating the construction of training courses using AI planning techniques. This would enable JITT. The feasibility of automated course construction has been demonstrated.
- The ILS could be embedded into larger systems. Within the ESSDE project, the ILS is seen as a facility to be embedded in the ESSDE toolkit. In particular, the ILS could be embedded into an operational user-performance system for OJT.

Learning Units

The LUDB contains the training knowledge for a given domain. There are two types of elementary knowledge:

- *Learning Element (LE).* An LE delivers a piece of training material designed to impart one or more skills.
- *Assessment Element (AE).* An AE tests whether or not the trainee has acquired a skill.

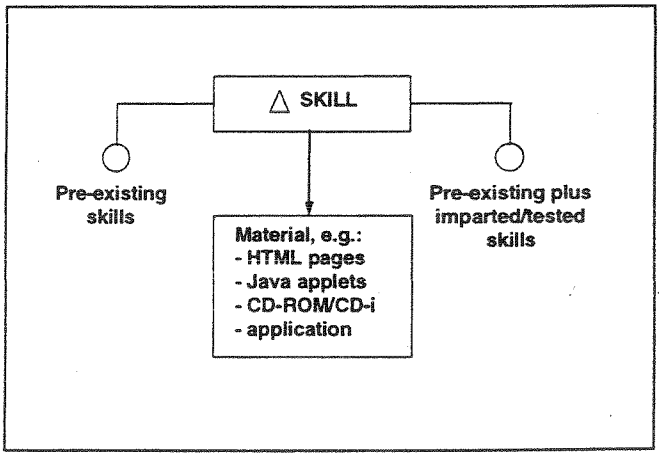


Figure 1. Schematic of Learning/Assessment Element.

The training material in LEs and the test material in AEs may be presented as web pages in HTML, Java or

[3] The correct upper and lower case is important.

Javascript code, (parts of) CD-ROM or CD-I presentations, (parts of) existing applications, etc. Figure 1 depicts an LE/AE schematically. Note that the LE or AE can be regarded as an indexing structure into the training/test material, which remains accessible (given the necessary permissions) for unguided browsing outside of the ILS.

A set of LEs and/or AEs related to one another is known as a *Learning Unit* (LU). Each LU is a linear sequence of one or more LEs and/or AEs. Usually, LUs consist of one or more LEs terminated by an AE, but other patterns are possible. In particular, an LU containing only AEs is known as an Assessment Unit (AU), and may be used:

- at the start of a training course to confirm that the trainee's existing skill-set subsumes the set of skills needed to assimilate the training material; and/or
- at the end of a training course to confirm that the trainee has acquired the skills needed to perform the task for which the course was designed.

The courseware designer determines the training and test material to be encapsulated in the LEs and AEs. The training and test material may well be pre-existing, as was the case for much of the ESSDE material.

The courseware designer also determines the grouping of LEs and AEs into LUs. There are organizational advantages in partitioning the training domain so that there is a one-to-one relationship between training topics and LUs. Each LU can be assigned an organizational owner who is responsible for keeping the training material in the LU up-to-date. To avoid the duplication of training material relating to any given topic, the ILS is designed so that an LU can be contained in multiple courses.

In addition to the LE/AE indexing structure, there are additional means in ILS for structuring the LUs including:

- grouping LUs into categories according to the Promesse role-task-skill model.
- allocating LUs to one of four intellectual skill classes (discrimate, apply-concepts, apply-rules, problem-solving).
- assigning LUs to one of four training base classes (process, methods, technical, domain-specific).

Inside the ILS, this indexing structure provides guidance in browsing the LUDB to facilitate refresher training. Moreover, the indexing structure is the key to the manual and automated construction of training courses.

Instruction Sequences

Training courses are represented as wholly- or partially-ordered sequences of LUs. Such an *instruction sequence* consists of a directed acyclic graph of LUs (see Figure 2).

A trainee may already have the skill knowledge that would be imparted by some of the LUs in the full instruction sequence. It would be both demotivating and wasteful of time and energy to teach these topics again. An instruction sequence can be tailored to deliver only those LUs the trainee really needs. Figure 2 depicts a case in which the trainee already knows items a, b and c and whose current task does not require him/her to complete the sequence.

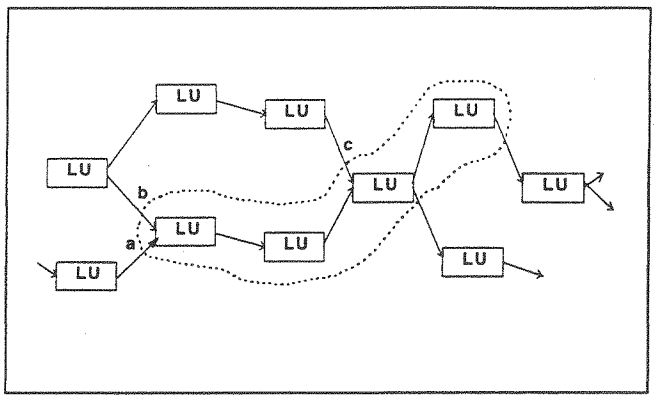


Figure 2. Instruction sequence as network of LUs.

Instruction sequences may be constructed either automatically using an AI planning algorithm or manually by an instructor or courseware designer. Automated construction has been demonstrated as a stand-alone application during Phase 2 of the ESSDE project. The currently-implemented PEE supports manual construction of instruction sequences by exploiting the knowledge in the indexing structure of LEs, AEs, and LUs. Mixed-initiative planning (i.e., the combination of automated and manual sequence construction) is under development in a separate, defence-related project; Grant (1997) describes a suitable graphic user-interface concept.

Comparison with other CBT frameworks

Other comparable CBT frameworks in European space applications include:

- *POINTER*. The Portable Intelligent Trainer for External Robotics (POINTER) uses expert systems techniques for domain knowledge and student modelling. POINTER has been applied to the European Robot Arm, to be flown on ISS. It is not Web-based and does not use AI planning techniques.
- *ASTRO*. The Advanced Software Tool for the tRaining of Operators (ASTRO) is a Web-based tool for distributing interactive multimedia training material (Wolff and Sanchez, 1996). ASTRO has been applied to the Tubular furnace with Integrated Thermal analysis Under Space conditions (TITUS). ASTRO does not use any AI techniques.
- *DMTS*. The Distributed Multimedia Training System enables instructors and students at geographically-separated locations to communicate in real-time by means of whiteboarding and video conferencing facilities. It is unclear whether DMTS uses Web-based technology; it does not use AI techniques.
- *ATIS*. Astronauts training: an intelligent system (ATIS) is a project to develop a knowledge-based system for supporting system and procedural self-

training of astronauts in the operation of payloads (Bertotti et al, 1995). The RAMSES payload, flown on the Spacelab IML-2 mission in 1992, serves as the illustrative application. Although ATIS uses expert systems techniques for domain and pedagogic knowledge and for student modelling, it does not appear to employ either AI planning or Web-based technology.

- *MOTE.* The Modular On-board Training Environment (MOTE) project is intended to prove that a CBT platform is well suited to changing environments, including off-nominal situations (Auferil and Bessone, 1997). Like ASTRO, MOTE uses the TITUS furnace as its illustrative application. MOTE is Web-based, but does not use any AI techniques.

The ILS is unique in combining Web and AI planning technologies for OJT and JITT in space applications.

AI Planning as Enabling Technology

In this chapter we justify the use of AI planning techniques as novel issues arise by comparison with operational applications. See (Grant, 1994) for technical details.

The first issue concerns the knowledge representation adopted. LEs and AEs are fundamental here; we consider LEs first. The purpose of LEs is to impart knowledge which the trainee can employ, i.e. to add to his/her repertoire of skills. The knowledge to be imparted is often known as the LE's *training objectives*. To achieve these objectives, the trainee must already possess certain other, more basic skills, often known as *training prerequisites*. Prerequisites and objectives can be expressed in terms of the trainee's skills. The prerequisites are the pre-conditions for delivering the training, and the post-conditions from delivering the training are the prerequisites plus the training objectives. This suggests that:

- LEs can be represented as planning operators, with pre- and post-conditions.
- the construction of instruction sequences can be represented as planning in a state-space expressed in terms of skills.

A second issue arises as to whether the instruction sequences are partially or fully ordered. We must consider execution to obtain an answer. Instruction sequences are intended to be delivered to trainees. Each trainee must undergo the complete instruction sequence. Moreover, each trainee can only do one thing at a time. Therefore, irrespective of whether the potential exists in an instruction sequence to deliver some LEs in parallel, the sequence must be linear when it is delivered. Hence, a linear plan-generation algorithm suffices.

Given that plan generation is state-based, uses operators with pre- and post-conditions, and outputs linear sequences, the STRIPS plan-generation algorithm (Fikes and Nilsson, 1971) is a reasonable choice. The classical planning assumptions embodied in STRIPS have been questioned, but, as McDermott (1994) points out, they are reasonable for many engineering applications.

The mapping from ILS to AI planning concepts is then as follows:

ILS concept	AI planning concept
Learning unit	Operator instance
Prerequisites	Operator pre-conditions
Training objectives	STRIPS-style add-list
Training prerequisites plus objectives	Post-conditions (a.k.a. effects)
Contents of LUDB	Operator-set
Trainee's existing skills	Current state (a.k.a. initial state)
Trainee's desired skills	Goal state
Instruction sequence	Plan
Instruction sequence construction	Plan generation

There remains a third issue. The imparting of knowledge by LEs would appear at first sight to be a monotonic process. Executed correctly, an LE adds skills to the trainee's store of knowledge and never removes them. In terms of the STRIPS representation, the LEs all have empty delete-lists. If so, AI planning techniques might seem to be an over-complication, with rule-based expert systems or decision tree techniques sufficing. Each LE could then be represented as a situation-action rule with the prerequisites as its antecedents and the training material as its consequents.

However, there are two factors which can rule out simple techniques:

- *Trainees can forget what they have previously learned.* Implicitly, there exist virtual operators with non-empty delete-lists additional to those representing the LEs. For obvious reasons, the operators of forgetting would not be included in instruction-sequence generation. To counter forgetting, an AU can be added at the start of the instruction sequence to check that the trainee has the necessary skills for assimilating the knowledge in the remainder of the instruction sequence. The AU would contain one AE for each skill in the prerequisites.
- *Trainees can fail to assimilate the training material.* Presenting an LE to the trainee does not guarantee that he/she will successfully assimilate the training material as skills. Although the LE's post-conditions would have been added to the modelled world state within the planner after the LE's execution, the same changes may not have occurred in the real world. To counter assimilation failure, an AU can be added at the end of the instruction sequence to confirm that the trainee has indeed successfully assimilated the training material in the instruction sequence. The AU would contain one AE for each skill in the training objectives.

In each case, the AU enables the training system to gain knowledge about the trainee's skills. In AI planning terms, the AU is needed to satisfy knowledge subgoals (Steel, 1991). This is easier to show for the case of countering assimilation failure. Suppose that a given LE imparts knowledge p (see Figure 3, in Steel's "tadpole" notation), then the AE that tests for p would have p in its pre-conditions and p V -p in its post-conditions[4]. The planning algorithm ensures that there is a *protection* for the condition p from the LE's post-conditions to the AE's pre-conditions, shown in the diagram as a horizontal line joining them. In the case of countering forgetting, the LE would have been (implicitly or explicitly) in some preceding instruction sequence.

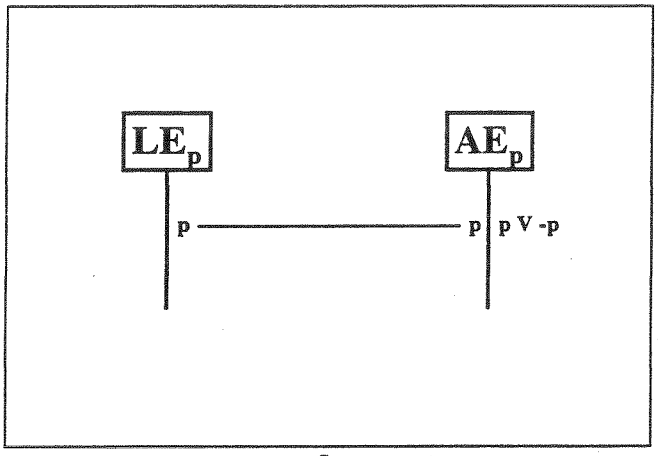


Figure 3. Dependency between LE and AE.

As the tadpole diagram shows, the world state after executing the AE may either contain the condition p or the condition -p. In AI planning terms, there are two possible worlds. The possible world containing -p is non-monotonic, because what was true (p) later becomes false (-p). Simple expert systems or decision-tree techniques are inadequate for constructing such instruction sequences, even though the non-monotonicity may only be a possibility. By contrast, a planning algorithm would insert into this possible world's instruction sequence another LE that re-asserts the condition p.

The exact state of the trainee's skills after executing the AE will only be known at execution-time. The crux of the problem is how to construct instruction sequences containing AEs. Possible solution approaches include:

- *Repeated replanning.* Construct the partial instruction sequence only up to the next AE. Execute this partial instruction sequence. Replan for the new initial state, and execute the new instruction sequence. Repeat until the training objectives are achieved. This approach depends on having a data-driven planning algorithm. However, most planning algorithms are goal-driven.
- *Optimistic planning.* Construct the instruction sequence assuming that all AEs will succeed. Execute the plan as long as the training objectives are confirmed. Handle negations in one of the following ways:
  - *Sudden death.* Terminate execution of the instruction sequence if the execution of any AE results in negation of a training objective. This approach has the disadvantage that the trainee's success is not guaranteed.
  - *Re-assertion.* Suspend execution of the instruction sequence, retrieve and execute a separate LE which will re-assert the negated training objective, and repeat the AE. This approach depends on providing the system with separate LEs for re-asserting all of the training objectives. The trainee's success within a finite time cannot guaranteed with this approach.
  - *Iterative planning.* Suspend execution of the instruction sequence, return the planner to the point in the plan just prior to an LE which asserts the negated training objective, and resume execution from that point. This approach depends on the segment of the instruction sequence preceding the AE containing an LE which asserts the negated training objective, as in Figure 3. Moreover, the planner must be *opportunistic*, in that it does not execute LEs for which the objectives have already been satisfied. The trainee's success within a finite time cannot guaranteed with this approach.
  - *Plan repair.* Suspend execution of the instruction sequence, repair the remainder of the instruction sequence for the new situation, and resume execution with the repaired instruction sequence. Plan repair is known to be a difficult problem.
- *Exhaustive conditional planning.* Construct the instruction sequences for all possible outcomes of the AEs. The instruction sequences can be merged into a single sequence with branch-points after each AE. At execution time, identify the correct branch to follow after executing each AE. This approach is impractical when there are many training objectives for the course, i.e., the instruction sequence has a high branching factor.

The ILS has been developed so as to support as many of these solution approaches as possible, with the courseware designer making the choice. Currently, the solution approaches must be executed manually, but could be automated by integrating planning into the ILS.

A further solution approach has also been implemented in the ILS. Within a given AE, the courseware designer can specify one of the following actions per attempt if the trainee fails:

- Repeat the test.
- Repeat the test, giving the trainee a hint.
- Note the failure, but continue the remainder of the instruction sequence as if the trainee had succeeded.

[4] The LE would be part of an LU, and the AE would be part of an AU.

For example, the test might be repeated on the first failed attempt, repeated with a hint on the second failed attempt, and the instruction sequence continued on the third failed attempt.

Application to ISS Training

Several authors have been considering ways of reducing International Space Station crew training time and increasing its effectiveness; see Muccio, Ockels and Gibson (1992), Griner, Lewis and Smith (1992), and Noneman (1992). Two options are relevant to this paper. The first option would be to increase training on scientific background and experiment objectives at the expense of procedural training. The second option would be to perform more training on-board.

Crew training on-board the ISS is a form of OJT. This could be implemented by incorporating training facilities in the crew terminals. Astronauts would then receive training using the same hardware and user interfaces that they use for operations. The third-generation Advanced Crew Terminal (Gale and Wolff, 1996), which is compliant with the ISS training hardware standards (ITCB, 1997), makes provision for a Simulation service for this reason. The ILS's learning units and instruction sequences can be structured by the courseware designer to be compliant with the ISS training software standards. Integration of a CBT framework with the Advanced Crew Terminal toolkit has been studied as a part of the POINTER-2 project (de Voogt, 1997). The ILS is one of the CBT frameworks under consideration, and is unique in combining AI planning and Web technologies (van der Arend, Kuiper and de Voogt, 1997). Moreover, the facility to integrate the ILS in a larger system can be used on the ground for the certification of crew members, planners, controllers, instructors, translators, Principal Investigators, and others who actively participate on console during a mission. This would be achieved by interfacing the ILS's LUDB and Knowledge Data Base to the European Astronaut Training Database and/or to the Training Administration and Management System.

The astronaut's mission will be several months long on the ISS. This means that the retention time expected of an astronaut since he/she was last trained on a payload on the ground is also of the order of months. Over this timescale, there is a substantial chance that payload planning will have changed. Even though the general rule on the ISS is that, if an astronaut has not trained for an operation on the ground then he/she does not perform the operation in-orbit, astronauts may still have to operate experiments for which his/her training is out-dated. This situation has already occurred on the MIR space station.

In particular, OJT aboard the ISS can reduce the retention time to a matter of days. Taking this trend further, the provision of JITT would minimize the time since the astronaut last trained on a payload. As its name suggests, JITT is intended to be delivered just prior to the moment when the knowledge gained is to be used. This has three benefits:

- The training effect is at its greatest, because the material is still fresh in the trainee's mind.
- Training is given only when it is needed, saving resources.
- The training material can be adapted to the precise circumstances in which the knowledge is to be used.

The ability of astronauts to react to unexpected situations is one of the key advantages of manned over unmanned spaceflight. On-board JITT can enhance these desirable capabilities of astronauts.

Each payload would be flown to the ISS together with its operating software, Virtual Control Panels, operating procedures, and training material (e.g. on a CD-ROM). The training material would be loaded on the crew terminals, as for the other payload-specific software.

The astronauts would consult the training material just prior to installing, commissioning, and using the payload. Although the crew activity plan would have to allow additional time for on-board training, the overall (i.e. space plus ground) training time would be reduced as a result of the heightened training effect. The heightened training effect would also increase the success rate in performing experiments, leading to an overall increase in the scientific utilization of the International Space Station.

Analogous arguments apply to the value of OJT and JITT for training ISS ground support personnel and users, as well as for crew training during Human Mars Exploration.

Conclusions

This paper has shown that operational support functions - most notably training of crew and ground support personnel - can benefit from advanced planning and scheduling technologies. The Integrated Learning System has been used as an illustrative example. The ILS is unique in that it combines AI planning techniques for constructing instruction sequences with Web-based training material. The techniques implemented in the ILS have been proven in the ESSDE application. Integration of the ILS with the third-generation Advanced Crew Terminal has been investigated so as to form a crew performance system with embedded OJT/JITT that is compliant with the ISS training standards for hardware and software. Embedding OJT in a crew performance system would reduce the retention time expected of astronauts from months to days. Adding JITT functionality, made possible by AI planning techniques, would minimise retention time and enhance the capability of astronauts to react to unexpected situations. The overall effect would be to increase the scientific utilization of the International Space Station.

References

Auferil, J., and Bessone, L. 1997. On-Board Training and Operational Support Tools Applying Web Technologies. *ESA Bulletin*, 89 (February).

Bertotti, L., Buciol, F., de Saint Vincent, A., Shafhouser, E., and Kass, J. 1995. ATIS - Astronauts training: an intelligent system.

de Voogt, E. J. 1997. Integration of CBT Framework and ACT. Netherlands Organisation for Applied Scientific Research (TNO), Physics and Electronics Laboratory, The Netherlands, version 0.3, 19 March 1997.

ESA. 1991. *European Space Agency Software Engineering standards*. ESA PSS-05-0, issue 2, 2 February 1991.

ESA 1993. *Columbus Software Development Standards*. EuroColumbus, document number STD 1213 800, issue 6, 25 June 1993.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence Journal*, 2, pp. 189-208.

Gale, L. P. 1993. Development of an Ada Strategy for Space Industries. In Proceedings of the 4th Ada in Aerospace Conference, Brussels, 8 November 1993.

Gale, L. P., and Mol, F. 1995. ESSDE Advanced Methods and Technologies, Phase 3: User Requirements Document, Integrated Learning System. Document D/4000/1, version 1.0, BSO/Origin Nieuwegein bv, The Netherlands, 18 August 1995.

Gale, L. P., and Wolff, M. 1996. Advanced Crew Terminal: A generic framework for the realisation of system and payload crew terminals. In Guyenne, T.-D. (ed). Proceedings of the 1996 Spacecraft Operations (SpaceOps'96) conference, Munich, Germany. ESA document SP-394, Volume 2, Track 4, paper SO96.4.19, ISBN 92-9092-259-1, pp. 713-720. Also available at URL: http://www.op.dlr.de/SpaceOps/spops96/miscmpp/mm-4-19/4_19.pdf.

Grant, T. J. 1994. ESSDE Advanced Methods and Technologies, Phase 2: Knowledge-Based Planning of Ada Training. Document 1707094, version 1.0, BSO/Origin Nieuwegein bv, The Netherlands, 26 September 1994.

Grant, T. J. 1997. A GUI Concept for Mixed-Initiative Planning. In Fox, M. (ed). Proceedings of the 16th workshop of the UK Planning and Scheduling Special Interest Group, Durham University, UK. Forthcoming.

Griner, C. S., Lewis, C. M., and Smith, K. A. 1992. Payload Training for the Space Station Era. In Proceedings of the 43rd Congress of the International Astronautical Federation, paper IAF-92-0706.

ITCB. 1997. Multilateral Training Management Plan, Volume 1. International Training Control Board, International Space Station Program, 1 September 1997

McDermott, D. 1994. The Current State of AI Planning Research. Invited paper. In Proceedings of the International Conference on Industrial and Engineering Applications of AI and Expert Systems.

Muccio, J., Ockels, W. J., and Gibson, E. 1992. Training for International Space Station 'Freedom' - A New Perspective. *ESA Bulletin*, 68, pp. 85-92.

Noneman, S. R. 1992. Ground Tended Payload Operations of Space Station Freedom. In Proceedings of the 43rd Congress of the International Astronautical Federation, paper IAF-92-0714.

Steel, S. 1991. Knowledge Subgoals in Plans. In J. Hertzberg (ed.). Proceedings of the first European Workshop on Planning, March 1991, Sankt Augustin, Germany. Lecture Notes in Artificial Intelligence, Vol. 522, Springer-Verlag, Berlin, Germany, ISBN 3-540-54364-3, pp. 112-121.

Van der Arend, J. G. M., Kuiper, H., and de Voogt, E. J. 1997. POINTER-2: CBT Survey Document (WP 1200). Netherlands Organisation for Applied Scientific Research (TNO), Physics and Electronics Laboratory, The Netherlands, version 1.1, 21 March 1997.

Wolff, M., and Sanchez, M-L. 1996. ASTRO: Computer-based payload operations training. ESA, *Preparing for the Future*, vol. 6, no. 4.

Active Statistical Learning of Heuristic Scheduling Strategies

Jonathan M. Gratch[2], Steve A. Chien[1], and Darren Mutz[1]

[1] Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA

[2] Information Sciences Institute, University of Southern California, Marina del Rey, CA

Email: {steve.chien,darren.mutz}@jpl.nasa.gov, gratch@isi.edu

Abstract

Although the general class of most scheduling problems is NP-hard, in practice, domain-specific techniques frequently solve problems in much better than exponential time. Unfortunately, constructing special-purpose systems is a knowledge-intensive and time-consuming process that requires a deep understanding of the domain and problem-solving architecture. In *adaptive problem-solving* the system automatically learns an effective domain-specific search strategy given a general problem solver with a flexible control architecture. In this approach, a learning system explores a space of possible heuristic methods for one well-suited to the eccentricities of the given domain and problem distribution. In this article, we discuss an application of the approach to scheduling satellite communications. Using problem distributions based on actual mission requirements, our approach identifies strategies that not only decrease the amount of CPU time required to produce schedules, but also increase the percentage of problems that are solvable within computational resource limitations.

1 Introduction

With the maturation of automated problem-solving research has come grudging abandonment of the search for "the" domain-independent problem solver. General problem-solving tasks like planning and scheduling are provably intractable. Although heuristic methods are effective in many practical situations, an ever growing body of work demonstrates the narrowness of specific heuristic strategies (e.g., (Baker, 1994, Frost & Dechter, 1994, Kambhampati, Knoblock & Yang, 1995, Stone, Veloso & Blythe, 1994, Yang & Murray, 1994)). Studies repeatedly show that a strategy that excels on one task can perform abysmally on others. These negative results do not entirely discredit domain-independent approaches, but suggest that considerable effort and expertise is required to find an acceptable combination of heuristic methods, a conjecture supported by the few published accounts of real-world implementations (e.g. (Wilkins, 1988)). The specificity of heuristic methods is especially troubling when we consider that problem-solving tasks frequently change over time. Thus, a heuristic problem solver may require expensive "tune-ups" as the character of the application changes. *Adaptive problem solving* is a general method for reducing the cost of developing and maintaining effective heuristic problem solvers. Rather than forcing a developer to choose a specific heuristic strategy, an adaptive problem solver adjusts itself to the idiosyncrasies of an application. This can be seen as a natural extension of the principle of least commitment (Sacerdoti, 1977). When solving a problem, one should not commit to a particular solution path until one has information to distinguish that path from the alternatives. Likewise, when faced with an entire distribution of problems, it makes sense to avoid committing to a particular heuristic strategy until one can make an informed decision on which strategy performs better on the distribution. An adaptive problem solver embodies a space of heuristic methods, and only settles on a particular combination of these methods after a period of adaptation, during which the system automatically acquires information about the particular distribution of problems associated with the intended application.

More rigorously, the adaptive problem–solving problem can be describes as follows. Given a flexible perfor-

mance element PE with control points $CP_1 ... CP_n$, where each control point CP_i corresponds to a particular control decision and for which there is a set of alternative decision methods $M_{i,1} ... M_{i,k}$,[1] a control strategy is a selection of a specific method for every control point (e.g., STRAT = $<M_{1,3}, M_{2,6}, M_{3,1}, ...>$). A control strategy determines the overall behavior of the scheduler. It may effect properties like computational efficiency or the quality of its solutions. Let PE(STRAT) be the problem solver operating under a particular control strategy. The function U(PE(STRAT), d) is a real valued utility function that is a measure of the goodness of the behavior of the scheduler over problem d. The goal of learning can be expressed as: given a problem distribution D, find STRAT so as to maximize the expected utility of PE. Expected utility is defined formally as:

$$\sum_{d \in D} U(PE(STRAT), d) \times probability(d)$$

For example, in a planning system such as PRODIGY [Minton88], when planning to achieve a goal, control points would be: how to select an operator to use to achieve the goal; how to select variable bindings to instantiate the operator; etc. A method for the operator choice control point might be a set of control rules to determine which operators to use to achieve various goals plus a default operator choice method. A strategy would be a set of control rules and default methods for every control point (e.g., one for operator choice, one for binding choice, etc.). Utility might be defined as a function of the time to construct a plan, cost to execute the plan, or some overall measure of the quality of the plan produced.

The adaptive problem–solving technique is applicable in cases where the following conditions apply:

1. The control strategy space can be structured to facilitate hillclimbing search. In general, the space of such strategies is so large as to make exhaustive search intractable. Adaptive problem–solving requires a transformation generator that structures this space into a sequence of search steps, with relatively few transformations at each step.

2. There is a large supply of representative training problems so that an adequate sampling of problems can be used to estimate expected utility for various control strategies.

3. Problems can be solved with a sufficiently low cost in resources so that estimating expected utility is feasible.

4. There is sufficient regularity in the domain such that the cost of learning a good strategy can be amortized over the gains in solving many problems.

Due to space limitations we now turn to a description of the DSN application domain and results. For further details on the statistical learning techniques we use for adaptive problem–solving the reader is referred to (Chien et al. 1995, Gratch & Chien 1996).

2 Results of Applying Adaptive Problem–solving to Deep Space Network Scheduling

In order to assess the viability of adaptive problem–solving to real–world scheduling problems, we have applied adaptive problem–solving to a testbed scheduler which solved Deep Space Network Antenna Allocation problems. In this section we describe a formulation of the Deep Space Network antenna allocation problem, and outline results of applying adaptive problem–solving to learning control heuristics for a scheduler which solves this problem.

The Deep Space Network (DSN) is a multi-national collection of ground-based radio antennas responsible for maintaining communications with research satellites and deep space probes. DSN Operations is responsible for scheduling communications for a large and growing number of spacecraft. This already complex scheduling problem is becoming more challenging each year as budgetary pressures limit the construction of new antennas. As a result, DSN Operations has turned increasingly towards intelligent scheduling techniques as a way of increasing the efficiency of network utilization. As part eof this ongoing effort, the Jet Propulsion Laboratory (JPL) has been given the responsibility of automating the scheduling of the 26-meter sub-net; a collection of

1. Note that a method may consist of smaller elements so that a method may be a set of control rules or a combination of heuristics. Note also that a method may also involve real–valued parameters. Hence, the number of methods for a control point may be infinite, and there may be an infinite number of strategies.

26-meter antennas at Goldstone, CA, Canberra, Australia and Madrid, Spain.

2.1 DSN Scheduling Problem

Scheduling the DSN 26-meter subnet can be viewed as a large constraint satisfaction problem. Each satellite has a set of constraints, called project requirements, that define its communication needs. A typical project specifies three generic requirements: the minimum and maximum number of communication events required in a fixed period of time; the minimum and maximum duration for these communication events; and the minimum and maximum allowable gap between communication events. For example, Nimbus-7, a meteorological satellite, must have at least four 15-minute communication slots per day, and these slots cannot be greater than five hours apart. Project requirements are determined by the project managers and tend to be invariant across the lifetime of the spacecraft.

In addition to project requirements, there are constraints associated with the various antennas. First, antennas are a limited resource – two satellites cannot communicate with a given antenna at the same time. Second, a satellite can only communicate with a given antenna at certain times, depending on when its orbit brings it within view of the antenna. Finally, antennas undergo routine maintenance and cannot communicate with any satellite during these times.

Scheduling is done on a weekly basis. A weekly scheduling problem is defined by three elements: (1) the set of satellites to be scheduled, (2) the constraints associated with each satellite, and (3) a set of *time periods* specifying all temporal intervals when a satellite can legally communicate with an antenna for that week. Each time period is a tuple specifying a satellite, a communication time interval, and an antenna, where (1) the time interval must satisfy the communication duration constraints for the satellite, (2) the satellite must be in view of the antenna during this interval. Antenna maintenance is treated as a project with time periods and constraints. Two time periods conflict if they use the same antenna and overlap in temporal extent. A valid schedule specifies a non-conflicting subset of all possible time periods where each project's requirements are satisfied.

The automated scheduler must generate schedules quickly as scheduling problems are frequently over-constrained (i.e. the project constraints combined with the allowable time periods produces a set of constraints which is unsatisfiable). When this occurs, DSN Operations must go through a complex cycle of negotiating with project managers to reduce their requirements. A goal of automated scheduling is to provide a system with relatively quick response time so that a human user may interact with the scheduler and perform "what if" reasoning to assist in this negotiation process. Ultimately, the goal is to automate this negotiation process as well, which will place even greater demands on scheduler response time (see (Chien & Gratch, 1994) for some preliminary work on this later problem). For these reasons, the focus of development is upon heuristic techniques that do not necessarily uncover the optimal schedule, but rather produce adequate schedules quickly.

2.2 Adaptive LR–26 Control Points

Pseudocode for the LR–26 scheduler is given below with the four control points indicated parenthetically. Control point one controls which partial schedule is chosen from the agenda, two determines in what order children in S are explored, three dictates which constraint is chosen to satisfy next, and control point four determines how schedules are refined. The details of how 1, 2, 3, and 4 are implemented define the runtime efficiency characteristics of the algorithm almost entirely (for a more detailed discussion see Gratch & Chien, 1996).

LR-26 Scheduler

Agenda := $\{S^0\}$;
While *Agenda* $\neq \emptyset$
(1) Select some partial schedule $S \in$ *Agenda*; *Agenda*:=*Agenda*–$\{S\}$
(2) Weight search for some $S^*(u) \in S$;
IF $S^*(u)$ satisfies the project requirements (*PR*) Then
Return $S^*(u)$;
Else
(3) Select constraint $c \in PR$ not satisfied by $S^*(u)$;
(4) Refine S into $\{S^i\}$, such that each $S^G \in S^i$ satisfies c
and $\cup\{S^i\} = S$;
Perform constraint propagation on each S^i
Agenda := *Agenda*$\cup\{S^i\}$;

Figure 1: The basic LR-26 refinement search method.

1)	Value ordering:	penalize-conflictedness
	Weight search:	first-solution
	Primary constraint ordering:	penalize-unforced-periods
	Secondary constraint ordering:	prefer-total-conflictedness
	Refinement method:	systematic-refinement
2)	Value ordering:	prefer-gain
	Weight search:	first-solution
	Primary constraint ordering:	penalize-unforced-periods
	Secondary constraint ordering:	prefer-total-conflictedness
	Refinement method:	systematic-refinement
3)	Value ordering:	penalize-conflictedness
	Weight search:	first-solution
	Primary constraint ordering:	penalize-unforced-periods
	Secondary constraint ordering:	penalize-satisfaction-distance
	Refinement method:	systematic-refinement

Figure 2: The three highest utility strategies learned by Adaptive LR-26.

2.3 Overall Results — DSN DISTRIBUTION

In this section we summarize the results of adaptive problem solving over the constructed DSN problem distribution. The results support that the system learned search control strategies that yielded a significant improvement in performance. Adaptive problem solving reduced the average time to solve a problem (or prove it unsatisfiable) from 80 to 40 seconds (a 50% improvement).

Due to the stochastic nature of the adaptive scheduler, different strategies were learned on different trials. All learned strategies produced at least some improvement in performance. The best of these strategies required only

24 seconds on average to solve a problem (an improvement of 70%). The fastest adaptations occurred early in the adaptation phase and performance improvements decreased steadily throughout. It took an average of 62 examples to adopt each transformation. Adaptive LR-26 showed some improvement over the non-adaptive scheduler in terms of the number of problems that could be solved (or proven unsatisfiable) within the resource bound. LR-26 was unable to solve 21% of the scheduling problems within the resource bound. One adaptive strategy substantially reduced this number to 3%.

In Figure 2 we give the three highest utility strategies learned by the adaptive algorithm. Most of the performance improvement (about one half) can be traced to modifications in LR-26's weight search method (i.e., how the algorithm chooses which child to explore next). The rest of the improvements are divided equally among changes to the heuristics for value ordering (the order in which partial schedules are prepended to the agenda), constraint selection, and refinement.

Another claim is that, in practice, adaptive problem–solving can identify strategies that rank highly when judged with respect to the whole strategy space. A secondary question is how well does the expert strategy perform. The improvements of Adaptive LR-26 are of little significance if the expert strategy performs worse than most strategies in the space. Alternatively, if the expert strategy is extremely good, its improvement is compelling.

As a way of assessing these claims we estimate the probability of selecting a high utility strategy given that we choose it randomly from one of three strategy spaces: the space of all possible strategies (expressible in the transformation grammar), the space of strategies produced by Adaptive LR-26, and the trivial space containing only the expert strategy. This corresponds to the problem of estimating a *probability density function* (p.d.f.) for each space: a p.d.f., $f(x)$, associated with a random variable gives the probability that an instance of the variable has value x. More specifically we want to estimate the density functions, $f_s(u)$, which is the probability of randomly selecting a strategy from space s that has expected utility u.

We use a non-parametric density estimation technique called the kernel method to estimate $f_s(u)$ (as in (Smyth, 1993)). To estimate the density function of the whole space, we randomly selected and tested thirty strategies. All of the learned strategies are used to estimate the density of the learned space. (In both cases, five percent of the data was withheld to estimate the bandwidth parameter used by the kernel method.) The p.d.f. associated with the single expert strategy is estimated using a normal model fit to the 1000 test examples from the previous evaluation.

2.3.1 *DSN Distribution*

Figure 3 illustrates the results for the DSN distribution. In this evaluation the learned strategies significantly outperformed the randomly selected strategies. Thus, one would have to select and test many strategies at random before finding one of comparable expected utility to one found by Adaptive LR-26. The results also indicate that the expert strategy is already a good strategy (as indicated by the relative positions of the peaks for the expert and random strategy distributions), indicating that the improvement due to Adaptive LR-26 is significant and non-trivial.

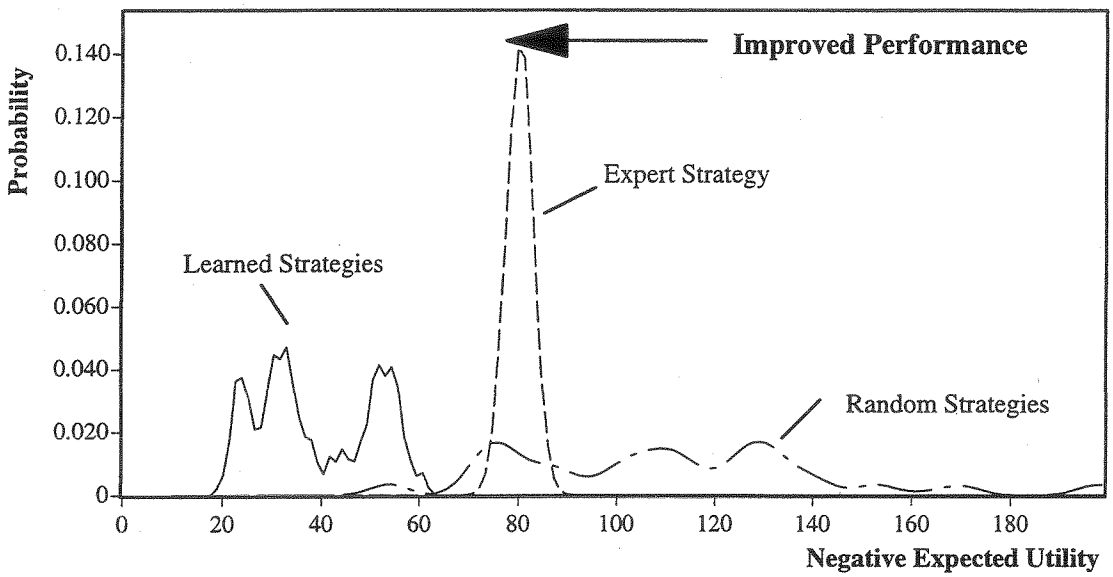


Figure 3: The DSN Distribution. The graph shows the probability of obtaining a strategy of a particular utility, given that it is chosen from (1) the set of all strategies, (2) the set of learned strategies, or (3) the expert strategy.

The results provide additional insight into Adaptive LR-26's learning behavior. That the p.d.f for the learned strategies contains several peaks, graphically illustrates that different local maxima exist for this problem. Thus, there may be benefit in running the system multiple times and choosing the best strategy. It also suggests that techniques designed to avoid local maxima would be beneficial.

2.3.2 *Full Augmented Distribution*

Figure 4 illustrates the results for the full augmented distribution. The full augmented distribution differs from the DSN distribution in that it also includes unsolvable problems posed to the scheduler (e.g., those problems requiring relaxation of project constraints or for which no known strategy was able to solve the problem). The results are similar to the DSN distribution: the learned strategies again outperformed the expert strategy which in turn again outperformed the randomly selected strategies. The data shows that the expert strategy is significantly better than randomly selected strategies. Together, these two evaluations support the claim that Adaptive LR-26 is selecting high performance strategies. Even though the expert strategy is quite good when compared with the complete strategy space, the adaptive algorithm is able to improve the expected problem solving performance.

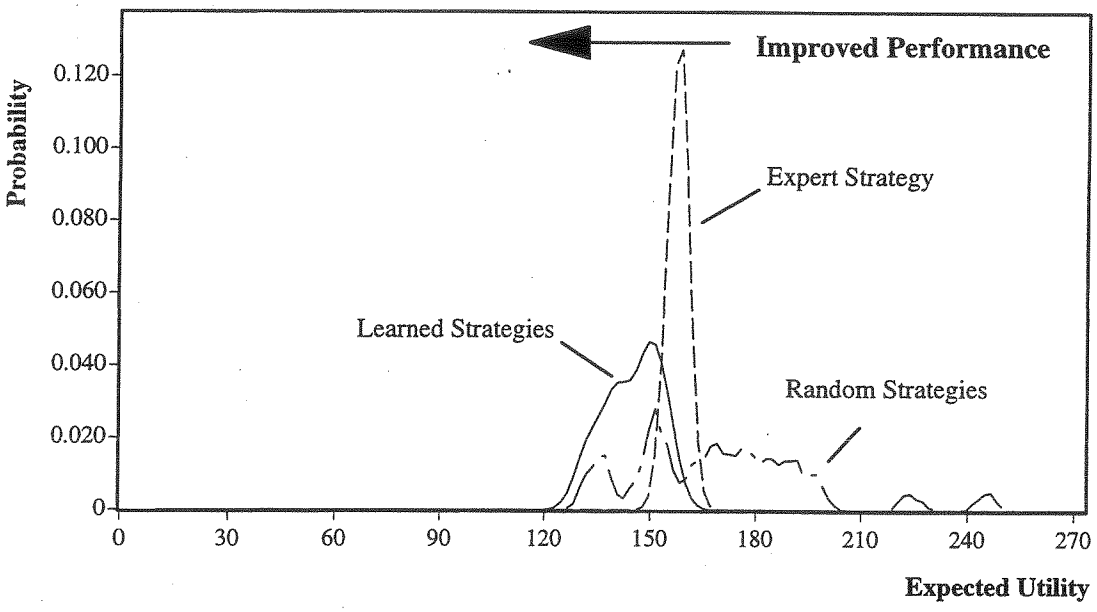


Figure 4: The Full augmented distribution. The graph shows the probability of obtaining a strategy of a particular utility, given that it is chosen from (1) the set of all strategies, (2) the set of learned strategies, or (3) the expert strategy.

3 Related Work and Conclusions

Related efforts include Heuristic–Biased Stochastic Sampling (HBSS) (Bresina, 1996) and GENH (Morris, Bresina, Rodgers, 1997). The approach described in this paper attempts to learn the optimal hypothesis to apply to a class of problems, based on a number of sample problem instances. HBSS and GENH differ in that they are designed to speed up search on individual problem instances.

Other related work includes more statistical techniques, which explicitly reason about performance of different heuristic strategies across a distribution of problems. These are generally statistical generate-and-test approaches that estimate the average performance of different heuristics from a random set of training examples, and explore an explicit space of heuristics with greedy search techniques. Examples of such systems are COMPOSER (Gratch & DeJong, 1992), PALO (Greiner & Jurisca, 1992), and the statistical component of MULTI-TAC (Minton, 1993). Similar approaches have also been investigated in the operations research community (Yakowitz & Lugosi, 1990). These techniques are easy to use, apply to a variety of domains and utility functions, and can provide strong statistical guarantees about their performance. They are limited, however, as they are computationally expensive, require many training examples to identify a strategy, and face problems with local maxima. Furthermore, they typically leave it to the user to conjecture the space of heuristic methods (see (Minton, 1993) for a notable exception).

This paper has described the application of statistical learning techniques to the refinement of heuristics for scheduling in an approach we call adaptive problem–solving. We first described the overall approach, in which the adaptive problem–solver searches through a space of potential control strategies for the scheduler. We then described results from applying adaptive problem–solving to a Deep Space Network Antenna Scheduing system. In this application, adaptive problem–solving was able to significantly improve upon the best human expert derived control strategy.

Acknowledgements

Portions of this work were performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration and portions at the Beckman Institute, University of Illinois under National Science Foundation Grant NSF–IRI–92–09394.

References

Baker, A. (1994). The Hazards of Fancy Backtracking. In *Proceedings of AAAI–94.*

Bell, C., & Gratch, J. (1993). Use of Lagrangian Relaxation and Machine Learning Techniques to Schedule Deep Space Network Data Transmissions. In *Proceedings of The 36th Joint National Meeting of the Operations Research Society of America, the Institute of Management Sciences.*

Biefeld, E., & Cooper, L. (1991). Bottleneck Identification Using Process Chronologies. In *Proceedings IJCAI–91.*

Bresina, J. (1996) Heuristic–Biased Stochastic Sampling. In *Proceedings of AAAI–96.*

Chien, S. & Gratch, J. (1994). Producing Satisficing Solutions to Scheduling Problems: An Iterative Constraint Relaxation Approach. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems.*

Chien, S., Gratch, J. & Burl, M. (1995). On the Efficient Allocation of Resources for Hypothesis Evaluation: A Statistical Approach. *Institute of Electrical and Electronics Engineers Transactions on Pattern Analysis and Machine Intelligence 17(7),* 652–665.

Dechter, R. & Pearl, J. (1987). Network–Based Heuristics for Constraint–Satisfaction Problems. *Artificial Intelligence 34(1)* 1–38.

Dechter, R. (1992). Constraint Networks. In *Encyclopedia of Artificial Intelligence*, Stuart C. Shapiro (ed.).

Fisher, M. (1981). The Lagrangian Relaxation Method for Solving Integer Programming Problems. *Management Science 27 (1)* 1–18.

Frost, D. & Dechter, R. (1994). In Search of the Best Constraint Satisfaction Search. In *Proceedings of AAAI–94.*

Gratch, J. & DeJong, G. (1992). COMPOSER: A Probabilistic Solution to the Utility Problem in Speed–up Learning. In *Proceedings of AAAI–92.*

Gratch, J., Chien, S., & DeJong, G. (1993). Learning Search Control Knowledge for Deep Space Network Scheduling. In *Proceedings of the Ninth International Conference on Machine Learning.*

Gratch, J., Chien, S. (1996) Adaptive Problem–solving for Large –Scale Scheduling Problems: A Case Study, Journal of Artificial Intelligence Research 4 pp. 365–396.

Gratch, J. & DeJong, G. (1996). A Decision–theoretic Approach to Adaptive Problem Solving. *Artificial Intelligence (Winter 1996).*

Greiner, R. & Jurisica, I. (1992). A Statistical Approach to Solving the EBL Utility Problem. In *Proceedings of AAAI–92.*

Kambhampati, S., Knoblock, C. & Yang, Q. (1995). Planning as Refinement Search: A Unified Framework for Evaluating Design Tradeoffs in Partial Order Planning. *Artificial Intelligence: Special Issue on Planning and Scheduling 66,* 167–238.

Kwak, N. & Schniederjans, M. (1987). *Introduction to Mathematical Programming*, New York: Robert E. Krieger Publishing.

Minton, S. (1993). Integrating Heuristics for Constraint Satisfaction Problems: A Case Study. In *Proceedings of AAAI–93.*

Mackworth, A. (1992). Constraint Satisfaction. In *Encyclopedia of Artificial Intelligence*, Stuart C. Shapiro (ed.).

Minton, S. (1988). *Learning Search Control Knowledge: An Explanation–Based Approach*, Norwell, MA: Kluwer Academic Publishers.

Morris, R., Bresina, J., & Rodgers, S. (1997) Automatic Generation of Heuristics for Scheduling. In *Proceedings of IJCAI–97.*

Sacerdoti, E. (1977). *A Structure for Plans and Behavior.* New York: American Elsevier.

Smyth, P. (1993). Probability Density Estimation and Local Basis Function Neural Networks. *Computational Learning Theory and Natural Learning Systems 2.*

Stone, P., Veloso, M., & Blythe, J. (1994). The Need for Different Domain–Independent Heuristics. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems.*

Taha, H. (1982). *Operations Research, an Introduction,* Macmillan Publishing Co.

Wilkins, D. (1988). *Practical Planning: Extending the Classical Artificial Intelligence Planning Paradigm.* San Mateo, CA: Morgan Kaufman.

Yakowitz, S. & Lugosi, E. (1990). Random Search in the Presence of Noise, with Application to Machine Learning. *Society for Industrial and Applied Mathematics Journal of Scientific and Statistical Computing 11, 4,* 702–712.

Yang, Q. & Murray, C. (1994). An Evaluation of the Temporal Coherence Heuristic for Partial–Order Planning. *Computational Intelligence 10, 2.*

Exploring AI Alternatives in Support of the General Observer Program for NGST

Sandy Grosvenor, Julie Breed, Lisa Dallas, Anuradha Koratkar

NASA/Goddard Space Flight Center
Code 522
Greenbelt, MD 20771
Sandy.Grosvenor@gsfc.nasa.gov
Julie.Breed@gsfc.nasa.gov
Lisa.Dallas@gsfc.nasa.gov
koratkar@stsci.edu

Abstract

One of the manually intensive efforts of the Hubble Space Telescope (HST) operations is the specification and validation of the detailed proposals submitted by scientists observing with HST. In order to meet the operational cost targets for the Next Generation Space Telescope (NGST), this process needs to be dramatically less time consuming and less costly. We are involved in evaluating this process and prototyping how artificial intelligence and user interface advances can be applied to reduce the time and effort involved for both scientists and the telescope operations staff. This paper describes the problem area more fully and reports on the current status and direction of the prototyping project.

Summary of Proposal Process with HST

HST uses a two-phase proposal process. In the first phase, scientists provide a description and scientific justification for the proposed observations along with preliminary information about targets and exposures. Phase I proposals are collected in an annual cycle. A peer review process reviews the submitted Phase I proposals and accepts proposals based on available time and scientific merit.

Once a Phase I proposal has been accepted the scientists must fill out a more detailed Phase II proposal that specifies the targets, instrument modes and exposure times in sufficient detail to allow HST's operations systems to schedule and execute the observations.

There are several tools that support the Phase II process. Many are available publicly, including the Remote Proposal Submission software (RPS2), an X-Windows based graphical user interface developed by the Space Telescope Science Institute (ST ScI) staff, as well as web-based reference manuals, and exposure calculators. Additional tools are available to the ST ScI staff.

NGST will be a fundamentally simpler observatory than HST. It will fewer instruments, the instruments will be simpler, and its location in space will provide fewer viewer constraints. These factors will help reduce the complexity of specifying of detailed observing proposals.

However, even greater gains in efficiency are needed in order to meet the operational cost targets for the Next Generation Space Telescope (NGST). The labor and costs involved in managing the general observer program still need to be lowered substantially.

Objectives of the Prototyping Phase

Our main objectives for development of the Scientist's Expert Assistant (SEA) are:

a) The system should be *intelligent.* It should employ artificial intelligence methodologies and paradigms to assist and guide the scientist in producing a proposal that is flight ready.

b) The system should be highly *intuitive.* The user interface should not require extensive training, scientists should be able to work with SEA with little or no assistance.

c) The system should be *distributed.* It should allow delivery and processing of proposals via the World Wide Web across a wide range of computing systems.

d) The system should be *adaptable.* As the telescope staff learns how best to use NGST once it is launched and operational, the system should be able to learn from that experience to further improve its effectiveness.

e) The system should be easily *integrated* with other NGST

planning and operations modules.

f) The system should be *flexible*. Since NGST is not scheduled for launch until about 2007, the system development must allow for changes in technology. Further, much of the process of developing observing proposals is common among observing platforms. This system could and should be an effective alternative for other observatories, both present and future.

We are currently looking at the overall process and how it can be better supported through automated means both in a general sense and in areas specific to NGST. Thus far, we believe the following will result in substantial improvements:

- Utilize expert assistant technology to conduct an on-line, guided interview with the general observer in order to solicit science requirements in "science language" and translate and package these requirements into a flight-ready proposal. The "interview" should accommodate a range of user types, from "novice" to "expert", and the system should be able to hide the minute details of the observatory from its user.

- Define and create an object-oriented system that provides "generic" support for a two-phase proposal development process. Then, develop framework-specific modules to support NGST, and also other observatories prior to NGST's launch.

- Utilize newer, more graphical and dynamic user interface techniques to make the process more intuitive.

- Better integrate reference material, providing an intelligent context-sensitive means to helping scientists find relevant technical reference material quickly and easily.

- Better integrate the various computational tools - for example, once the instrument mode and targets have been selected, the scientist should not have to go to a separate exposure calculator tool, re-enter that information to obtain exposure times, and manually enter those exposure times back into the proposal.

- Standardize the interface for the computational tools across the different instruments in order to decrease the learning curve and increase efficiency.

The current plan is for the SEA to be explored first through two prototyping and "proof-of-concept" phases to evaluate how well the goals can be met. If successful, the next step is to build the operational SEA for an instrument on an operational observatory in order to develop operational experience with the SEA. This will ensure that by the time the instruments for NGST are well defined, the SEA will be a proven platform that simply needs adapting to the NGST specific instruments.

Goddard's Advance Architectures and Automations Branch is working with the STScI to explore SEA alternatives with the goal of using the Advanced Camera for Surveys (ACS) instrument (scheduled to be installed in HST in 1999) as our operational test-bed.

Progress to Date

Thus far, the team has focused on gaining an overview of the proposal process and defining the scope and schedule for the prototype effort. We have reviewed the overall Phase II process and the areas within in that are currently manually intensive. We have also begun developing the rules for prototyping an instrument configuration module for ACS. Finally, we are investigating similar development efforts for other observing platforms to see if we can leverage their work.

In the process of our analysis, we feel that there are five priority areas for development. These modules will initially be developed separately, but we expect that the tools will be integrated into a single iterative user-interface:

Graphical, "real-time" exposure calculator

This initial tool will generate real-time interactive graphs showing Signal-Noise Ratio and Source counts across a range of exposure times and wavelengths. The tool will allow the scientist to edit target or instrument parameters and instantly see their effect. We are targeting a Beta release for ACS by the end of December 1997. This tool is being developed in Java and will be a fairly simple application that should be able to replace the need to browse through over 100 pages of graphs and tables typically found in the HST Instrument Manuals. The development of this tool will also provide an initial development platform for the user-interface guidelines and underlying data objects that will be used for subsequent tools. We expect that in the first phase, this tool will not use expert system technology.

"Visual" Target Tuner (VTT)

The VTT is a graphical tool for fine tuning target coordinates and orientation. Currently, observers must independently research target information and manually enter the information into their proposal. If they have a need to include or exclude specific objects, they must manually determine a specific orientation for the instrument. Giving a precise orientation requirement significantly hampers the schedulability of their program.

The VTT will know, however, the areas that need to be included or excluded and can therefore pass on to the scheduling system and range of acceptable orientations. Further, there are currently no visual tools to help predict the overlap of spectroscopic slits, or the impact of refraction spikes.

The VTT seeks to be that visual environment. We are planning to prototype the VTT in several phases. The first phase (targeted for mid-1998) will be limited to displaying a previous FITS image, allowing the user to specify inclusion or exclusion areas, and fine tune the specific location. In the second phase (targeted for late-1998) we will add the ability to model defraction spikes and spectroscopic slits. In both phases, we anticipate that this tool will be primarily a visual and graphical aid.

Instrument Configuration Expert System (ICES)

This module will be a rule-based expert system that will guide the observer through the definition of instrument parameters by asking a series of science-based questions, and then providing recommended settings for the instrument based on the answers received. The goal will be to eliminate the need for the observer to study and absorb the range of technical details about the workings of an instrument, and instead let them focus on the science they want to achieve. This tool will also be developed in several phases. The first phase will have a fairly small rule-system that will focus on filter selection, and will emphasize the development of a good user-interface system and standards.

The user-interface for this expert system has some relatively unique needs. It needs to be able to integrate with the other tools and therefore have a compatible look and feel. It needs to be able to ask and save responses to questions in a manner that will be acceptable to both advanced and novice users. It needs to transparently interact with both the user on the "front-end" and a rules-engine on the "back-end". It also must support intelligent cross-references to technical literature, since while we are trying to allow observers to bypass up-front study of the technical instrument parameters, we are *not* trying to prevent them from studying the technical details. We want to help them focus quickly on the areas that are most relevant to their science objectives.

The second phase of the ICES will concentrate on expanding the rules and capabilities of the system. This is a critical objective for the SEA. We are striving to discover if the tool can contain a sufficient level of science expertise to free the observers from the technical details of the instrument and significantly reduce the support needed from Institute experts. We also must find out once such a system is achieved, if can we gain the acceptance from the observing community.

Visit Planner Expert System (VPES)

Thus far the modules described will primarily focus on defining a single exposure, both target parameters through the VTT and instrument parameters through the ICES. The Visit Planner Expert System will work to provide assistance in laying out multi-exposure "visits". Both observing scientists and Institute staff currently spend a great deal of time planning multi-exposure visits. These challenges include:

- Laying out multiple exposures to create a mosaic
- Imaging a single target with a variety of instrument configurations.
- Planning not just the individual exposure times, but also the overhead time necessary to perform other tasks such as slewing the telescope, and reading the CCD buffers after an exposure.

These are currently manual, iterative processes that involved balancing exposure times to achieve the desired science objectives while keeping within the overall visit time constraints. The VPES will be an expert system that will query the observer with a series of questions about their science objectives and priorities. It will be able to recommend an optimal trade-off between individual exposure times and total visit execution time.

Re-validation Agent

This module has not yet been fully analyzed. Currently, the Program Coordinators at the Institute spend a great deal of time re-processing already approved, but still pending proposals when a change to the instrument occurs. These changes can include a variety of things, for example, new calibration information that affects optimal exposure times. This concept for the re-validation agent is to use agent-based technology to evaluate the impact of changes to both submitted proposals and proposals that are still under development. The agent could seek out impacted proposals, calculate the effect of the impact, develop a recommendation and then notify the observer and Institute staff of possible alternatives.

Issues and Challenges

We have several challenges to overcome in developing this prototype. This section will highlight a few of those.

The first challenge is to gain synergy with other development efforts. We recognize that we are not the only group interested in improving resources for observers. The Gemini group is developing a new system, as is the Very Large Telescope (VLT) group. We need to stay in touch

with these efforts and develop synergy where possible to minimize development costs and maximize product quality.

Second, we want the system to be applicable to other observing platforms. While space-based astronomy has a significant set of unique issues and parameters, the fundamental process of defining an observing proposal is not particularly unique between observing platforms. We are striving to develop a system that will be easily adaptable to different instruments, different target types, and to different base observatories.

Third, we are faced with rapidly changing and evolving technology. While this prototyping effort will use a "near-term" instrument, ACS as the testbed, the long-term objectives are cost-effective systems for NGST, a system that is not scheduled to launch for almost ten years. The technological changes over the last ten years make it likely that not only will the basic technology we use today be obsolete, but the whole approach system may have been replaced by a system that is not yet even in the conceptual thinking of most developers. Consequently, we are involved in a trade-off between developing with proven established methods and emerging technologies and approaches. Our currently target environment is Internet-based and platform-independent. We are using object-oriented design with Java as the development language. We are balancing between the expectation that computing and network speeds will continue to dramatically improve and working with the existing limitations.

Finally, we are striving to achieve a balance between solving problems that may be unique to HST (both its hardware and its orbital constraints) while conceiving of what different constraints there may be for NGST.

Conclusion

We have sent representatives to this Workshop on Planning and Scheduling for a variety of reasons. We want to share our research to date. We want to seek out additional input on approaches and methodologies. We are interested in generating interest in our project. And we are looking for ways in which our work might provide a basis to help other observing platforms reduce the costs of their proposing systems.

Old and New Ideas for Integrating Planning and Execution

Othar Hansson, Jordan M. Hayes, Charles A. Ocheret

Thinkbank, Inc.
1678 Shattuck Avenue, Suite 320, Berkeley, CA 94709-1631
othar@thinkbank.com, jordan@thinkbank.com, chuck@thinkbank.com

Abstract

We sketch some features of the planner/executive interface in SchedKit, a toolkit of reusable software components for science planning and spacecraft sequencing. Our interest in execution was spurred by lessons from the New Millenium Remote Agent (NMRA) project. Many other computer scientists have also studied the control of execution in complex, uncertain environments: here we describe elementary concepts of *transaction-processing* that have inspired our design. Finally, we describe some novel modeling techniques supported by SchedKit.

Planning and Execution

Thinkbank, Inc. is developing SchedKit, a toolkit of reusable software components for science planning and spacecraft sequencing. This paper describes the planner/executive interface within SchedKit. Our design has been influenced by the New Millenium Remote Agent (NMRA), currently being implemented at NASA Ames and at JPL for the DS-1 asteroid rendezvous mission. Details of NMRA can be found in recent papers by Pell et al. (96a,96b,97a,97b) and Muscettola et al. (97).

We are particularly interested in the interface between the planner and the executive. How does the planner plan for execution? How should the two modules divide up the labor: how much reasoning and constraint processing should be done at run-time? What "virtual machine" should the executive present to the planner? Is there a single representation of plans that meets the needs of planning and execution?

Historically, planners and schedulers have not taken seriously the problems of execution. This dates back to the earliest efforts at optimizing operations. For example, one of the 19th-century ancestors of modern operations research was Frederick Taylor: "Taylorism" or scientific management was rightly criticized for placing all the focus on planning and scheduling (i.e., management functions), and for assuming that flawless and repeatable execution (by human laborers) was possible. The vain attempt to separate cognition from action is recapitulated in modern operations research and artificial intelligence. As a result, the brittleness of schedules still limits the adoption of scheduling technology in factories, transportation systems, space missions, and other application domains.

Much recent work in AI has attempted to integrate planning and execution. And the NMRA team has highlighted the importance of integrating planning and execution, if a scheduler is to be flown on-board a spacecraft: in fact, Pell et al. (97a) claim that "NMRA is one of the first systems to integrate closed-loop planning and execution of concurrent temporal plans."

NMRA consists of three main modules: The Planner/Scheduler (PS), the Executive (EXEC) and the Mode-Identification and Recovery system (MIR). Loosely speaking, PS generates episodic plans, passes them to EXEC for execution, and waits for the planning horizon to expire or for MIR or EXEC to detect failure conditions. MIR has its own lower-level rapid recovery planner as well. MIR and EXEC handle some errors, but when a plan failure occurs, PS is reinvoked to generate a plan to patch in at a rendezvous point in the future.

In theory, PS can produce more robust schedules than traditional schedulers, because PS produces least-commitment schedules, with flexible time-windows for each spacecraft activity. That this is a revolutionary idea is perhaps reflected in the widespread use of "sequence" to describe the output of NASA schedulers and planners: a sequence is a total order, but PS produces plans that are partial orders with some run-time flexibility. In other words, each activity is not assigned a precise time for execution: the executive can choose a time consistent with a set of simple temporal constraints (difference constraints). At the expense of added software complexity, this least-commitment approach should allow execution to continue even after deviations from predicted behavior.

The Planning/Execution Interface: PRL

In designing SchedKit, we have the luxury of starting with a clean sheet of paper in designing the planner, the executive, and the interface between them. In contrast, NMRA began with relatively standard architectures for the planner (based on HSTS) and the executive (based on RAPS), and designed the planner/executive interface by adjusting existing inputs and outputs. This was of course necessary to meet mission schedules and reduce risk.

If, however, we start from scratch, one way to approach the design problem is to pursue an "execution-as-computation" or "computer language" analogy.

Specifically, the plan representation language (PRL) is viewed as the instruction set of a virtual machine; the executive is the runtime system for this virtual machine; and the planner is the compiler. The planner compiles sets of goals (the source language) into plans (the target language, i.e., the plan representation language), which are then executed (by the executive, which implements the target language virtual machine). The activities in the plans are operations made available by the virtual machine.

There are some notable constraints to the design problem. The planner's inputs (the source language) are constrained by the demands of the application. In the case of a spacecraft, the source language must be simple but flexible to meet the demands of ground controllers. Likewise, the executive's outputs are tied to the spacecraft hardware. But importantly, the PRL interface between planner and executive is unconstrained.

There are many choices in designing the PRL. By making the virtual machine more powerful, we can replace the executive's behavior and logic by corresponding statements in the plan representation language. Conversely, we can make the plan representation language trivial, i.e., equivalent to the source language, thus passing all of the planner's "normal" responsibility onto the executive. This is a well-examined tradeoff in the design of computer instruction sets ("reduced" versus "complex" instruction sets, i.e., RISC versus CISC). A low-level PRL (analogous to a RISC design) allows for a very simple and fast executive, but requires more work on the part of the planner (or compiler).

The PRL choice in NMRA was made according to the following principle: "In DS1, activities are abstracted to the level where there are no interactions among their subactivities. This level allows the planner to resolve all of the global interactions without getting into details that would over-constrain the executive." (Pell et al. 97b). Over-constraining EXEC would increase the risk of plan failure. PRL should also specify failure modes and recovery alternatives: in NMRA, recovery semantics seem to be represented entirely within EXEC, and not within the "plans" themselves.

Virtual Machine example: Transactions

In designing SchedKit, we sought to borrow as much technology as possible from other areas of computer science. Transactions are a central abstraction used in implementing modern distributed systems. As Gray and Reuter (93) put it, "In a nutshell: without transactions, distributed systems cannot be made to work for typical real-life applications." And yet transactions arose not as a naturally evolved abstraction or requirement, but as a radical proposal to overcome the chaos of distributed systems implementation projects at IBM and elsewhere.

The "ACID" properties define transactions:

- Atomicity

  transactions have "all-or-nothing" effects, and will undo on abort.
- Consistency

  integrity constraints are maintained.
- Isolation

  each transaction executes as if unaffected by concurrently executing transactions.
- Durability

  after a system failure, the achieved state of the system is recovered automatically.

Through the ACID properties, a database management system provides a virtual machine: in other words, it guarantees certain behavior that is abstracted away from the uncertainty or other details of actual execution.

We will briefly compare transaction-processing applications to "typical" planning/scheduling systems. Both need atomicity because failures are unavoidable, and application designers demand some simple and universal recovery mechanism. Ad hoc recovery mechanisms, with unique guarantees, result in chaos.

Durability deals with even more sophisticated error recovery: ensuring that the achieved state of the system is re-created after a failure (part of MIR's role). But in a planning/scheduling application, some aspects of the system state may not be controllable, while others may be irrelevant to the remainder of the plan. In short, only the preconditions for the remainder of the plan need to be durable in the face of failures. Finding such a satisfactory recovery state after failure may involve search. But without some durability mechanism, execution halts at the first global error that was not anticipated in the plan itself (e.g., as a contingent execution path).

Consistency is a responsibility that is presently diffused among the plan itself, EXEC, and MIR: we advocate collecting and exposing these integrity constraints. The final ACID property is isolation: activities may be easier to specify if we can isolate them from the immediate effects of concurrently-executing activities. Isolation is partially achieved in planner/schedulers by careful specification of preconditions and "parameter bindings" within activities. After parameters are bound and preconditions checked, the activity is typically oblivious to changes in parameters and preconditions.

In some senses, the demands of planning and scheduling are more difficult than transaction-processing. But if our systems lack the fundamental ACID properties such as atomicity and consistency, then it will be as difficult to debug planning applications as it is to debug distributed systems without transactions. It may be that entirely different abstract properties or "guarantees" will prove to be more suitable. But the ACID properties are a natural starting point, and we hope to prove or disprove their usefulness in the applications we prototype and develop using SchedKit.

Execution Support in SchedKit

A few of SchedKit's features are directly borrowed from the transaction-processing framework (Gray and Reuter 93, Papadimitriou 86). By using these familiar guarantees provided by transaction-processing systems, we hope to simplify the task of designing and debugging planning/scheduling applications. Still other features are borrowed from the distributed systems "protocol design and validation" literature (e.g., Holzmann 91).

Activities in a SchedKit plan are atomic in the sense that failure modes and recovery actions can be specified. Failure modes are simply represented as preconditions for the corresponding recovery actions. Global plan failures can be monitored by activities inserted into the plan. These failure models are incorporated in the plan by use of a simple task decomposition (i.e., they are automatically included as part of the expansion of a higher-level activity). Consistency of global integrity constraints can be checked similarly: an integrity constraint is an activity that is always ready-to-run, but whose preconditions are triggered only when an integrity constraint is violated. Finally, as a durability mechanism, we hope to be able to use an existing recovery system.

One major gap that we see in SchedKit and NMRA (and other systems) is a lack of a metric for execution performance. What exactly is the goal of the executive in a particular domain? An executive which "fails aggressively" produces safe but unproductive results. An executive which "recovers aggressively" risks spacecraft health. We feel that such tradeoffs can only be addressed by explicit modeling: each failure and recovery should somehow be annotated with its associated risks. Utility theory is one promising modeling tool available to place this planning/execution tradeoff on a sound footing. We hope to extend SchedKit's modeling language in this direction, based on some of our previous decision theory work (Hansson 97).

One final point on SchedKit's design is the use of parameterization throughout the system. Wherever possible, decisions are reified or represented as explicit constrained variables. For example, in modeling the constraints of the domain (analogous to NMRA's Domain Description Language), we use Modeling variables and named constraints to make design assumptions explicit ("if Modeling(EffectX) then Enforcing (ConstraintY)"). We are experimenting with a similar approach to represent abstraction levels. A final example of parameterization is within the search algorithms: branch points, algorithm parameters, etc. are reified as variables to aid in explanation.

For further details on SchedKit, visit our web-site at http://www.thinkbank.com.

Acknowledgments Thanks to many members of the NMRA team at NASA Ames and JPL for helpful discussions. This work is supported by NASA Contract NAS2-97008.

References

[Gray and Reuter 93] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann, San Mateo, CA.

[Hansson 97] Othar Hansson. Bayesian Problem-Solving Applied to Scheduling. Doctoral dissertation, University of California (Berkeley), forthcoming.

[Holzmann 91] Gerald J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, Englewood Cliffs, NJ.

[Muscettola et al. 97] Nicola Muscettola, Chuck Fry, Kanna Rajan et al. On-Board Planning for New Millenium Deep Space One Autonomy. In *Proceedings of IEEE Aerospace Conference*, Snowmass, CO.

[Papadimitriou 86] Christos Papadimitriou. *The Theory of Database Concurrency Control.* Computer Science Press, Rockville, MD.

[Pell et al. 96a] Barney Pell, Douglas E. Bernard, Steven A. Chien, Erann Gat, Nicola Muscettola, P. Pandurang Nayak, Michael D. Wagner, and Brian C. Williams. A Remote Agent Prototype for Spacecraft Autonomy. In *Proceedings of the SPIE Conference on Optical Science, Engineering, and Instrumentation.*

[Pell et al. 96b] Barney Pell, Erann Gat, Ron Keesing, Nicola Muscettola, and Ben Smith. Plan Execution for Autonomous Spacecraft. Appears in the *Working Notes of the AAAI Fall Symposium on Plan Execution,* Cambridge, MA, 1996.

[Pell et al. 97a] Barney Pell, Douglas E. Bernard, Steven A. Chien, Erann Gat, Nicola Muscettola, P. Pandurang Nayak, Michael D. Wagner, and Brian C. Williams. An Autonomous Spacecraft Agent Prototype. In *Proceedings of the First International Conference on Autonomous Agents*, Marina del Rey, CA 1997.

[Pell et al. 97b] Barney Pell, Erann Gat, Ron Keesing, Nicola Muscettola, and Ben Smith. Robust Periodic Planning and Execution for Autonomous Spacecraft. *Proceedings of IJCAI-97.*

Using Common Graphics Paradigms
Implemented in a Java Applet
to
Represent Complex Scheduling Requirements

by John Jaap, Elizabeth Davis, & Patrick Meyer
Mission Planning Division
Mission Operations Laboratory
George C. Marshall Space Flight Center
National Aeronautics and Space Administration

Abstract

The experiments planned for the International Space Station promise to be complex, lengthy and diverse. The scarcity of space station resources will cause significant competition for resources between experiments. The scheduling job facing the Space Station mission planning-software requires a concise and comprehensive description of the experiments' requirements (to ensure a valid schedule) and a good description of the experiments' flexibilities (to effectively utilize available resources). In addition, the continuous operation of the station, the wide geographic dispersion of station users, and the budgetary pressure to reduce operations manpower make a low-cost solution mandatory. A graphical representation of the scheduling requirements for station payloads implemented via an Internet-based application promises to be an elegant solution that addresses all of these issues.

Scheduling Requirements

For activities on the International Space Station, scheduling requirements are described by defining "activities" and "sequences of activities." Activities generally equate to the simplest or lowest-level tasks. A sequence of activities is usually required to represent scheduling entities. Consider the following example from everyday life of an Internet user. To check e-mail from home, one must perform a sequence of three activities: connect to an Internet Service Provider (ISP), download new messages, and terminate the modem connection. The "scheduling entity" is the sequence of three tasks. Doing the activities out of sequence or standalone does not accomplish the objective.

Activities define the resource requirements (with alternatives) and other quantitative constraints of tasks to be performed. In the e-mail example, resource usage (e.g., phone line, ISP account and mail program) is requested by the activities. Of course, there may be alternative resources; one may have multiple phone lines, multiple ISPs and multiple mail-client programs. Even the computer itself is a resource that may be shared with other household members. The resource requirements for these activities are shown in the following table.

Connect	Download	Disconnect
Phone line	Phone line	Phone line
ISP account	ISP account	ISP account
Computer	Computer	Computer
	Mail Client	

While this representation of the activities seems straightforward, it may not be complete. The "Connect" activity would look more like the following if we show alternative resources.

Connect
- or { Home-office phone / Residence phone
- or { Employer's ISP / AOL
- Computer

If there is some hierarchical relationship between the choices, we might show the activity the following manner.

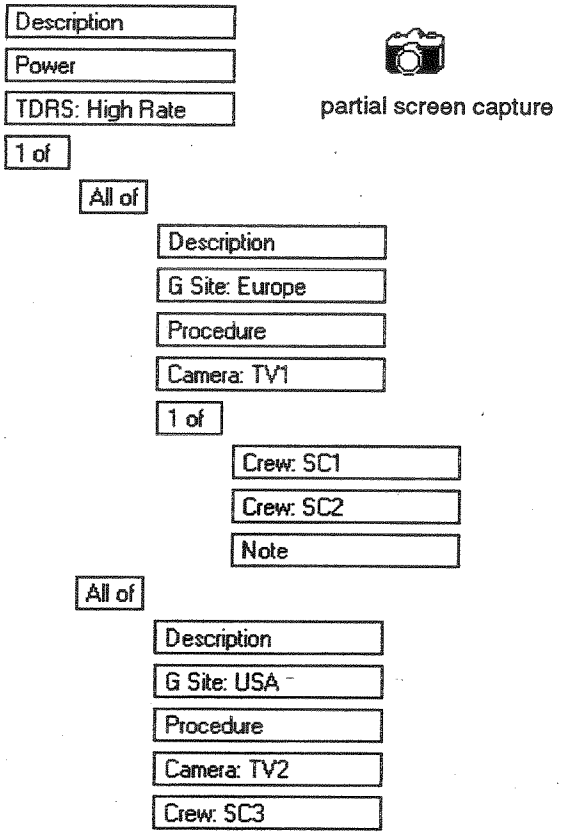
Connect
- or { and { Home-office phone / Employer's ISP } / and { Residence phone / AOL }
- Computer
- Electricity

Or, we might use the following "outline" paradigm.

Connect
- One of
  - All of
    - Home-office phone
    - Employer's ISP
  - All of
    - Residence phone
    - AOL
- Computer
- Electricity

The "outline" paradigm supports unlimited depth and can be intuitively manipulated by a drag-and-drop interface. The actual implementation of this method of representing activities uses dialog boxes that are activated by clicking on the constraints to enter constraint values.

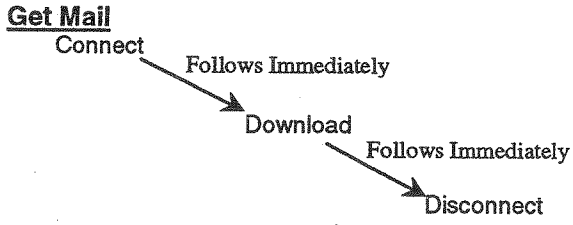
The figure below shows an example of an activity that might be flown on the space station.



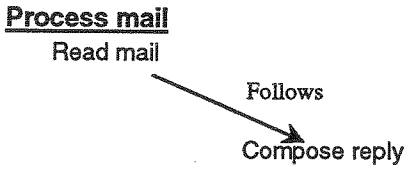
In this example, power and high-rate TDRS are always needed, and the activity must be performed over Europe or the USA. If done over Europe, a specified procedure is used to operate camera 1 by either crewman SC1 or SC2. If done over the USA, then a different procedure is used by crewman SC3 to operate camera 2. Of course the details of each constraint (for example, the amount of power) are entered into a dialog panel that appears when a box is clicked with the mouse. This example also shows that ancillary information such as procedures and descriptions can be associated with the constraint hierarchy.

For space station, the users are not allowed to create resource definitions; they are created by an administrator in negotiation with the station systems experts and with the user community. When defining an activity the user selects from a list of predetermined constraints. The dialog boxes for the values are tailored to the type of constraint. For space station, ten constraint types (and thirteen dialog panels) are currently defined.

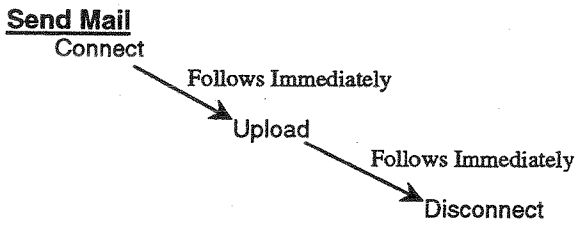
Sequences define the time relationships between activities. In our e-mail example we discussed three activities which would be done one after the other (i.e., in a sequence). This can be represented pictorially something like the following.



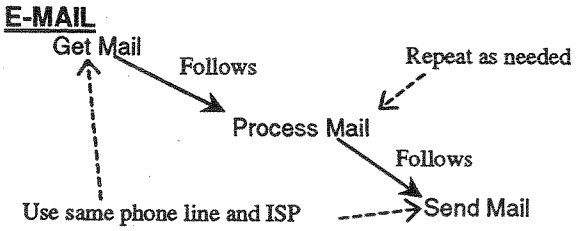
Consider two more sequences, one for processing mail (reading and composing replies to the mail) and another for uploading the replies to the ISP. To support these sequences three new activities (read mail, compose reply, and upload) are assumed but are not defined. The sequence for reading each message and composing a reply looks like the following.



The sequence for sending mail looks like the following.



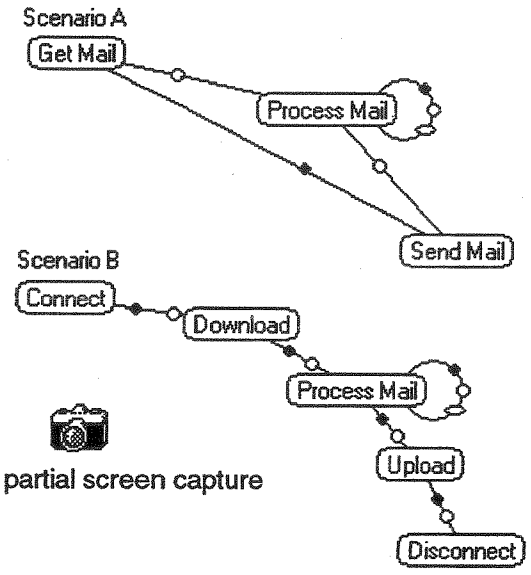
The whole process of getting mail, processing, and uploading mail would be expressed in a meta-sequence like the following.



In addition to the main sequence (Get Mail, Process Mail, Send Mail), instructions that multiple messages are to be processed and the same ISP and phone line are to be used for getting and sending mail are shown.

Of course, there is another scenario for doing one's e-mail; one could connect, download the mail, process it

(read and reply), upload and then disconnect. The implementation allows the user to define multiple scenarios of a sequence on the same drawing panel. Below is the actual representation of the e-mail sequence. Notice that both scenarios are shown and that sequences can be nested (Process Mail is a sequence) and mixed with activities.



The program shows the relationships between the items by showing a placeholder (circle for temporal relationship, oval for repetition relationship, or spot for resource persistence). The details of each relationship are entered via dialog panels that appear when a relationship placeholder is clicked with the mouse. Often, the relationship is more complex than simply "follows" of the e-mail example. A sequence may also show temporal relationships to station events which are not part of the experiment. For example, a payload activity or sequence might have a temporal relationship to a station event like reboost or shuttle docking.

Implementation

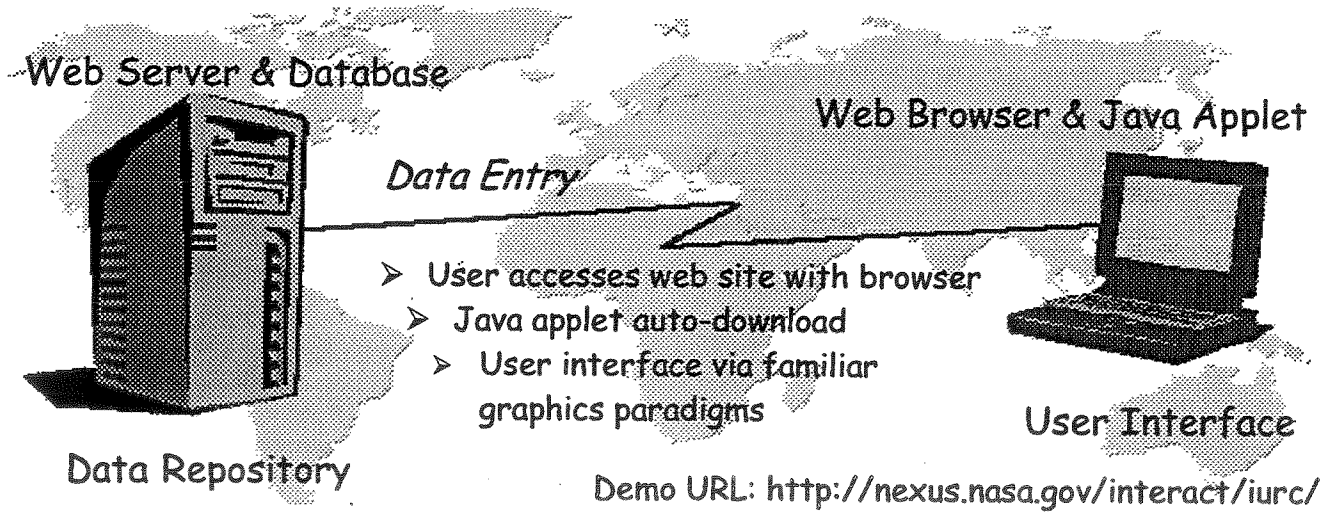
In support of space station, the Mission Planning Division at the Marshall Space Flight Center is implementing a Java-based remote data entry system to collect user requirements for payload planning and scheduling.

The data collection system has several components as shown in the figure below. The web server and the database are located at Marshall. A user connected to the Internet supplies a web browser. When the user points his browser to the web site, a Java applet is downloaded and executed (after user authentication). The applet provides the graphics and dialog interfaces which allow the user to enter/edit his scheduling requirements, and it uploads and downloads the requirements from the database on the web server.

Summary

In the e-mail example, we saw that simple graphics paradigms can encapsulate a wealth of information about hierarchies and relationships. However, using graphics paradigms to represent scheduling requirements has even greater advantages. Graphics paradigms are less susceptible to misinterpretation and can overcome language barriers; they can represent requirements concisely while capturing scheduling flexibility; and, with the recent advances in technology, they can be effectively implemented.

A Java applet, to run in a web browser, has been developed to support the graphical representation of payload scheduling requirements. Implementing the entry and editing of requirements via the web solves the problems introduced by the geographic dispersion of users. Reducing manpower is accomplished by developing a concise representation which eliminates the misunderstanding possible with verbose representations and which captures the complete requirements and flexibility of the experiments.



Modification of an Existing Planning and Scheduling System to a New Mission

Carl A. Johnson

Space Telescope Science Institute
3700 San Martin Drive
Baltimore, Maryland 21218
Cjohnson@stsci.edu

Abstract

The Space Telescope Science Institute (STScI) is currently participating in the development of the planning and scheduling system for the Far Ultraviolet Spectroscopic Explorer (FUSE) Mission at Johns Hopkins University (JHU). FUSE, one of the first of the Explorer class missions, appears to be a model for how future Explorer missions will rely on organizations with experience in developing planning and scheduling systems to offset their limited budgets, resources and development time. The FUSE project is an excellent example in demonstrating the process by which portions of an existing planning and scheduling system are modified to meet the different operational requirements and interfaces of a new spacecraft and instrument. This paper examines the experience gained from participating in such a process. It highlights some of the elements of the science mission planning process, addresses some lessons learned to date and describes areas where planning and scheduling software needs to be flexible.

Background

The Far Ultraviolet Spectroscopic Explorer is a PI-class NASA astronomy mission that will explore the Universe through high-resolution (lambda/delta lambda = 24,000-30,000) spectroscopy at far ultraviolet wavelengths (905-1195Å. FUSE is scheduled as a three-year mission within the NASA Origins program.

The FUSE satellite is composed of the spacecraft and the scientific instrument. The instrument consists of four co-aligned telescope mirrors (~ 39 cm x 35 cm clear aperture). The light from the four optical channels is dispersed by four spherical, aberration-corrected holographic diffraction gratings, and recorded by two delay-line micro-channel plate detectors. Two channels with SiC coatings cover the range 905-1100Å and two channels with LiF coatings cover the range 1000-1195Å. A Delta II vehicle will launch FUSE into an 800 km, 25-degree inclination orbit, from Cape Canaveral in the September of 1998.

The mission has both a Principal Investigator (PI) and Guest Investigator program with a PI observing time allocation of 60%, 40% and 25% respectively for the first three years. The GI submission process is a two- phase approach. In phase one, GIs provide preliminary targets and scientific justification suitable for an allocation committee to select programs. In phase two, GIs provide the targets and observation information needed by the science planning team for scheduling of the observations. The PI submission process only requires the above mentioned phase two portion with the PI teams handling their own target allocations. A representative science program developed from the science objectives contains approximately 1,400 observations of some 1,000 objects with cumulative on-target exposure times ranging from several minutes to more than 50 hours. Based on its duration, an observation may be broken into smaller observations to provide scheduling flexibility.

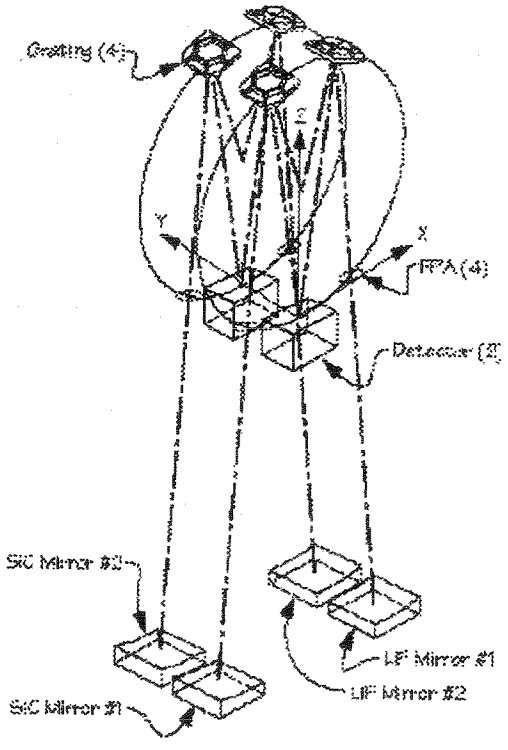


Figure 1. FUSE Instrument Diagram

Among other elements in its contract, STScI was selected to provide system engineering support and planning and scheduling software for FUSE. The system

engineering responsibilities include the definition of any new scheduling requirements, refinement of the operations plan, and support of other science operations requirements definition and development tasks. The planning and scheduling software to be delivered from STScI is a version of Spike that takes into account FUSE mission scheduling requirements.

Spike is a general system for planning and scheduling developed at STScI under contract with NASA. Spike has a full set of features to support planning and scheduling that includes a powerful yet efficient method (suitability functions) of representing a wide variety of strict and preferential constraints, absolute constraints, relative constraints, resource constraints, and a Constraint Satisfaction Problem (CSP) based scheduler.

Objective

The design and development of a science mission planning system is a non-trivial task. The objective of this paper is to provide some lessons that the development of the FUSE science operations system has revealed during its continued progress. Some of the lessons learned are an expression of the reality of the FUSE projects and other past experiences that may also pertain to your projects. The hope is that by remembering some of the information presented here, your development may be more successful.

FUSE Science Operations Concept

The FUSE science mission planning concept is similar to the one used by HST and other observer based satellites. As mentioned above, it uses a two-phase proposal approach. The first phase of the proposal is the submission of requests to receive observation time on the instrument. Potential observers submit the ASCII text version of a LaTeX proposal form electronically to the Guest Investigator program at Goddard Space Flight Center (GSFC). At the time of receipt, a message of acknowledgment is sent back to the submitter assigning the proposal a unique identifier. Should a problem be found later, an additional e-mail message will be sent with more details. Proposals can be re-submitted by placing their unique identifier in the subject line. A received proposal is saved to a file and a backup copy is sent to the FUSE project to provide feasibility analysis for the Target Allocation Committee (TAC). The TAC meets and determines proposals or portions of proposals that are to be allocated time. Observers are notified on the results of the TAC. Accepted programs are now required to submit a completed phase 2 form.

Phase 2 of the operations concept is the submission and scheduling of the accepted programs and begins with the receipt of the completed phase 2 forms from observers. The phase 2 form is an ASCII file, submitted electronically, that uses a set of keywords to define targets and observations. This form is parsed and its information is loaded into a relational database. The science planning team evaluates each observation and assigns them a particular observation type. This value and the information loaded into the database from the phase 2 form is used to expand the observation into the actual data elements used by Spike for long range planning and short term scheduling. The science mission planning team uses Spike as a long range planning tool to organize observations into one week scheduling bins. The completed long range plan is baselined, putting information on when a particular observation is to be scheduled back into the database for use in determining observations' field and guide stars. The science mission planning team uses Spike and a long-range plan to generate short-term schedules for portions, nominally one day, of a long range bin. These schedules identify the sequence of activities needed to complete an observation along with any activities required to move from one observation to the next. Short term schedules produce an ASCII Mission Planning Schedule(MPS) file that details the activities, events, and tables in the format needed by the Spacecraft Control Center(SCC) to generate spacecraft command loads. The MPS files are baselined to assure that they have been generated using the approved long range plan for their time frame and to pass more detailed information about the observations into the database for use by science data processing.

The above science mission planning concept has several areas worthy of highlighting. The first is the decision of the science mission planning teams to review each observation and identify it as a particular observation type. This decision was made to simplify the amount of information required by an observer in their phase 2 form and to give the team more flexibility in determining how an observation was to be scheduled. The science mission planning team felt that this method would minimize the need for observers to understand the details of how the spacecraft, instrument, and scheduling worked and allow them to concentrate on the type of science needed. Also of note is that only electronic submission of phase 1 and phase 2 forms are accepted.

A highlight of the FUSE design is an application called the Exploder. In HST Operations, proposals are broken down to exposure level data before being stored in a planning and scheduling database. The initial granularity for generating database inputs was at the proposal level making it difficult to regenerate data for an individual observation without re-running the entire proposal. FUSE operations learned from this and elected to load the proposal into the database before any action to manipulate the data was taken. The Exploder is responsible for converting observation information from the phase 2 form, the observation type supplied by the science mission planning team, observation type related rules, observation type related algorithms, and observation level overrides to generate the data needed by Spike for its long rang planning and short term scheduling. Using a technique learned from OPUS, the Exploder stores its rules and overrides directly in the database. This allows changes to

how the Exploder will expand information to be modified without the need of a build of the software. The algorithms, essentially rules that are too complex to be implemented using the rules method, are defined in the database. Elements of that definition are the order in which the algorithm should be called relative to other algorithms for a particular observation type and a list of parameter inputs to be read into the algorithms. Though not as flexible as the rules, existing algorithms can be added for a particular observation type via the database and defined input parameters can have their values changed without a software build. This feature, especially in the rules definitions, will allow FUSE to add and change rules as they become known during early operations. Another feature of the Exploder is that its granularity is at the observation level. The program allows the user to specify program, observation, list of observations, or entire database for the Exploder to create Spike input data, however, the program processes these an observation at a time. The benefit to this feature is that it allows science operations to regenerate Spike input data for an observation should its scheduling information change without affecting other observations in that observation's program.

Another highlight in the FUSE design is that Spike has the responsibility for breaking observations into its lower level exposures. In the HST scenario, all observations and exposures were defined before planning and scheduling software developed an actual schedule. In the FUSE Spike, the breaking of observations into exposures is pushed pack to where all the orbital information and non-observing overheads are known. This allows Spike to optimize the observation for any given orbit relative to the overall exposure duration of the observation. In addition, Spike has the ability to increase or reduce this overall exposure duration to improve efficiency. The end result is a schedule where the observation and its exposure duration take advantage of every second of available orbit time. To improve efficiency even more, most activities are considered time relative. This allows the spacecraft to start an activity earlier if previous activities have taken less time than expected. This works well for activities, like acquisitions and peakups, whose duration may be difficult to estimate at the Spike planning and scheduling phase.

Lessons Learned

In any project, hindsight is a wonderful thing. When looking back, there are often a number of things that you would do differently if given the opportunity to go back. Below are some of the things I would do differently or at least be prepared when they occur.

Staffing Profiles

Lesson 1. Consider the maturity of the subsystem or software to be developed against the overall maturity of the project when identifying your expected staffing profiles and delivery dates.

When costing out the FUSE proposal, we allocated staff in a uniform and smooth profile. This was done to keep the progress of work on a nice steady pace and to prevent the shuffling of developers between projects. The hope was that enough requirements would flow from the FUSE project to keep our staff busy. Two realities quickly set in. First, since the Spike software was a mature product, the level of information that our system needed was initially higher than what the FUSE project could provide. The project had not developed to the stage that it could give answers. An example of this was ground station contacts, we wanted to know how ground station contact information and its formats was to be passed to Spike while FUSE was still trying to figure out where and what type of ground station they were going to purchase. Second, the limited number of staff members forced each of them to carry a number of different responsibilities. This created a situation where work on any given task tended to come in bursts and where some activities needed to be pushed back due to staffing constraints. This put us in a situation where not only was the project not prepared to field our questions, they didn't always have the staff to track the answers down.

For the case where your subsystem is more mature than the overall system, you may need to adjust your staffing profiles to fit the growth pattern of the project. Recognize in advance that much of the information that you need will come in at the last possible moment and adjust you staffing profile to handle this situation. It may seem that you can use the information needs of your system to drive development. This may not work and may only serve to alienate. Nevertheless, not pushing at all can make it difficult for you to make your deadlines. The solution may lie in the middle ground, using your system to help keep focus on the information that need to be gathered and the design decisions that need to be made.

Operations Concept Definition

Lesson 2. Develop your science operations concept to a level of detail that allows you to validate your design and implementation decisions.

The high-level definition of a science operations concept is generally a data product presented in the Preliminary Design Review (PDR). In the case of an orbiting observatory, this concept presents the steps that a particular observation would flow through the system to get scheduled on the spacecraft. For the PDR, staying high level is a result of the overall amount of information covered at the review and the available time of the reviewers. A pitfall is that the concept does not get evaluated and reviewed at a more detailed level. This causes problems later on in the mission when holes are found or when it is discovered that several people had different impressions of what the concept really is.

Sample Observation Scenarios

Lesson 3. Define a list of the types of observations required for your mission and continue to expand their detail throughout your design and development phases.

An excellent tool for validating your science operations concept and design is the development of science observation scenarios. These scenarios begin as descriptions of observations that will be required of the spacecraft and then become more refined as the operations software develops. Initially, visualizing how a particular observation will flow through the system to the spacecraft will uncover potential flaws in the system. This is especially true for the rarer types of observations (i.e. moving target, bright target requiring acquisition followed by an offset) that may require additional instructions to be passed to the spacecraft than that required for a nominal observation.

As your development continues, the scenarios will be expanded to include the actual inputs needed for each piece of your science operations system. For example, as the definition of the phase one and phase two formats are defined, putting the scenarios in the selected format will validate the structure and identify any missing keywords. Later these can be used to actually test your phase one and phase two parsing software.

Re-Validating Existing Algorithms

Lesson 4. Review the algorithms in the existing code and validate them against the new operations concept, instrument and spacecraft.

A new mission means a new operations concept, a new instrument, and a new spacecraft. Any of these may invalidate an algorithm in an existing software product. The HST version of Spike assumed that its orbit was circular and therefore could use simpler orbit propagation models. In the HST operations concept, this is acceptable because a later software application (SPSS) not used by FUSE has more detailed ephemeris information. For FUSE, Spike needs to be able to handle the situation where the orbit may not be circular and thus needs to have more refined orbit interpolation algorithms. The validation of Spike against the operations concept allowed this problem to be found and resolved with minimal impact. Spacecraft slewing is example where a different spacecraft may invalidate an existing algorithm. The FUSE spacecraft uses eigenaxis slewing while HST uses great circle slewing. By catching these mismatches early in the development phase, FUSE was able to modify its version of Spike. These two examples serve to highlight that no matter how robust an existing application is, early and thorough review of its algorithms and design against the characteristics of the new mission will save headaches down the road.

Software Flexibility

The schedule of the FUSE project is such that detailed information on how the spacecraft and the instrument works will not be available until after much of the software has already been developed. Planning and scheduling software must be developed to minimize the impact of this situation. One technique used is to move operational information into table loads that can be stored in the database. This allows software to be developed using a standard method independent of the actual data that will follow. During actual operations, this technique allows the software to respond to changes without the need to recompile and redeliver code. Using the database to store the information allows a measure of configuration control. Some examples where this technique is used for FUSE are: The software that reads and loads the completed phase 2 forms into the database; The rule definitions used by the Exploder; The Algorithm order and input parameters used by the Exploder; and the definition of MPS activity formats generated by Spike.

The FUSE science planning system uses existing software, for example Spike, that already has a set of defined input and output data formats. Whenever possible, these formats were adopted or an effort was made to minimize the changes to an existing format. For the cases where formats existed for both sides of an interface, one method was chosen over the other. For example, though Spike currently has a format for reading in ground station contacts, the format from the LEO-T ground station software was selected. The reasons for this selection were that FUSE had less control over the LEO-T format and the likelihood of Spike using the LEO-T format on a later project. In developing Spike, acceptable old formats were not thrown away as other formats were added. The idea is that as Spike incorporates the common formats of these products it will become more robust.

Summary

The FUSE project has demonstrated that it is cost effective to adapt and apply existing planning and scheduling software applications to a new mission. The benefits are that it allows a new project a cheaper, faster and better mechanism for the development of its operational software.

Two features that resulted from the incorporation of STScI's existing planning and scheduling software into the FUSE science operations concept are:

- The incorporation of features implemented by OPUS into a new application called the Exploder that creates the Spike input products via phase 2 program data, observation types, rules, and algorithms.
- The technique of putting information into the database rather than directly into an application, thereby allowing modification without software builds.
- Pushing the breaking of observation into exposure into Spike to allow further optimization of orbit time.

- Making activities time relative to further allow improvement in efficiency of scheduling after it is loaded on board the spacecraft.

The incorporation of existing planning and scheduling software into the FUSE science operations concept to date has identified some lessons that can be applied to other missions. The are:

1. Consider the maturity of the subsystem or software to be developed against the overall maturity of the project when identifying your expected staffing profiles and delivery dates.
2. Develop your science operations concept to a level of detail that allows you to validate your design and implementation decisions.
3. Define a list of the types of observations required for your mission and continue to expand their detail throughout your design and development phases.
4. Review the algorithms in the existing code and validate them against the new operations concept, instrument and spacecraft.

For further project that incorporate existing software products, it is important to remember that improving the flexibility of your software now may make it easier later. Two areas where flexibility improvements can be realized are database usage and standardize inputs/outputs. For the database, creating mechanisms to pull in program information allows quicker turn around times for mission specific changes. Further, if these relations are designed with other projects in mind, they make it only necessary to modify data in the database to bring an application on-line. This has been demonstrated by OPUS. Standardizing inputs and outputs to match up with the software products that applications will most likely interface with will improve the plug-in capabilities of that application. For example, it is very likely that Explorer missions will predominantly use ground stations so by incorporating the ground station contact file format from LEO-T, Spike will support this system without any modifications.

References

Web Related Sites

The Restructured FUSE Mission, Science and Technical Forum: Revision 1, February 1996
fuse.pha.jhu.edu/pubinfo/docs_pub/scitech/scitech.html

Spike: Planning and Scheduling,
presto.stsci.edu/spike/

FUSE Mission Planning Database Design Document,
www.pst.stsci.edu/~cjohnson/fuse_db_design.html

FUSE Exploder Design Requirements,
www.pst.stsci.edu/~cjohnson/exploder_design.html

OPUS: An Operational Pipeline Unified System
samvax.stsci.edu/rose/opus/index.html

Rescheduling in Support of Space Operations

Steven Jowers

The Boeing Company
13100 Space Center Blvd.
Houston, TX
Steven.Jowers@boeing.com

Abstract

This paper provides a brief overview of the rescheduling problem from the perspective of an interval-based, non-chronological scheduler followed by a discussion of how we have solved the rescheduling problem. It will conclude with examples taken from complex aircraft assembly line operations, from a complex, resource rich test facility, and a discussion of how significant aspects of these problems are similar to ones encountered in space operations.

Introduction

Often our attention is focused on crafting an initial schedule, but once that schedule goes into actual operation, a different, but just as large a problem comes into existence - that of maintaining the schedule as it undergoes the stresses and strains of real-time operations. Besides being completed, tasks are delayed, only partially completed, deleted, and split into one or more new tasks as each day passes. This can quickly render the original schedule obsolete. Rescheduling for continued operations must consider how to manage these perturbations. Can we rebuild the entire schedule from a specified day forward, or can we perturb only parts of the schedule? Is it worth the time and effort to re-optimize? How can we track whether or not the perturbations are causing us to slip schedule? Has the set of critical resources changed? Should additional resources, if possible, be made available? Finally, any solution answering these questions must scale to very large, complex schedules.

The Rescheduling Problem

The problem domains needing a scheduler to support operations are quite varied. Test facilities, small payload operations, and simple on-orbit maintenance activities can all call out tasks with resource requirements and few, if any, predecessor/successor (precedence) constraints. Other problems, such as payload operations, complex maintenance operations, and on-orbit assembly operations will all have relationships between tasks such as precedence, relational (task to task), and temporal (task to time point) in addition to complex resource requirements.

Support for space operations necessitates scheduling of ground facilities (training and operations), mission planning prior to flight, and mission operations during flight. For space systems with long duration missions (e.g., the international space station), the distinction between mission planning prior to flight and mission operations during flight is quite blurred; once the mission starts there is no a priori "mission planning" prior to flight for everything anymore. Instead, the vehicle is in continuous mission operations. In this latter case, a capability to update an already executing schedule to reflect work actually accomplished and real-world, unplanned opportunities with possibly new constraints, and capability to insert significant new pieces into an already executing schedule (plans for new experiments, maintenance activities, etc.) is a must.

When generating the original schedule prior to actual operations, it must be guaranteed feasible with respect to all documented plan input data. Once in execution, however, real-world demands must allow for the selective relaxation/enforcement of constraints; exploration of hypothetical "what if" scenarios, oversubscription of resource constraints normally kept a conservative level but having additional reserve for emergencies, etc.

Once a schedule begins execution, tasks are hopefully performed according to schedule. Experience shows, however, that external factors cause the delay of some tasks (e.g., unexpected lack of power), the execution of tasks out of sequence (e.g., unexpected opportunities, delays incurred by predecessor tasks, etc.). Additionally, tasks may only be partially complete at reschedule time and their durations may have deviated from planned values.

The challenge presented to a scheduling engine builder whose product must support scheduling prior to and during operations is multifaceted. For our work, we have chosen to approach the problem from three vantages:

- development of a rich and extensible vocabulary for modeling task constraints, including task status data
- development of a scheduling engine for the data model
- and the development of controls for the scheduling process that allows users to execute their own heuristics

Modeling Vocabulary

It is not the intent of this paper to describe all the

constraints that our experience dictates should be in a scheduler for complex problems. Nor is it the intent to justify the statement that the easy addition of new constraints to the vocabulary is highly desirable. However, by providing a partial listing of constraints we believe need support, we can useful provide backdrop for later discussions. The constraint vocabulary includes:

- precedence constraints
- duration
- relational and temporal constraints
- numerous earliest start times/latest completion times
- resource expressions (arbitrarily deep nested ANDed and ORed resource requirements)
- resource reservation/provision, consumption/production
- multiple priorities
- preferences
- interruptability
- exclusive reservation of a resource to a group of tasks; i.e., resource lockout.
- task group membership where a group impacts resource and timeline availability for non-group members

These are just some of the constraints that the vocabulary should support. In addition to these, each task must allow for documenting the task's status. This is critical to the rescheduling process. If not provided as an element of the vocabulary, it is meaningless to talk of tracking a task's status and executing re-scheduling operations to account for that status.

Scheduling Engine Design Considerations

The scheduling engine builders face a number of hart design issues. They should design a scheduling engine which, at its core, doesn't need to do anything different for tasks having or not having status information. This simplifies complexity of the core engine, helps preserve performance since the same routines are used for all scheduling functions regardless of status information, and relegates the need for understanding the concepts associated with task status to high level data abstractions. It also helps to minimize the number of assumptions coded into the core engine.

Further, builders need to design algorithms such that operations which are not linear have their impact mitigated. Ideally, the time to (re) schedule should increase linearly with data set size.

The time to (re) schedule should be sufficiently fast to support operational considerations; i.e., it is impossible to use a solution for daily operations when the engine needs more than 24 hours of computation time to generate a schedule. For that matter, a significantly faster solution is required to allow exploration of "what-if" scenarios.

The scheduling engine should be capable of handling very large data sets. Scheduling in support of space operations for programs such as the International Space Station will require the scheduler to process many, many tasks against limited resource availability; e.g., maintenance operations, payload operations, on-orbit training, etc.

Finally, the scheduling engine's implementation should support ease of integration with other and perhaps older electronic data systems. This is critically important if there is an existing Information Systems infrastructure that must be utilized to support schedule execution; e.g., systems that log task status.

Control of the Scheduling Process

The existence of a rich modeling vocabulary and an engine which can process it is only part if the overall solution. Real-world concerns dictate that users must be given controls for the scheduling process itself. Different data sets will exhibit different complexities. The process of scheduling for one data set, which yields good results, may not do so for another data set. Consider the following examples:

1) Provide sufficient control over the scheduling process so those tasks with latest completion times, if possible, are scheduled without constraint violations; i.e., schedule backward from the documented completion times.
2) Support allocation of a specific resource to a defined group of tasks exclusively once that group has been started. Don't allow any tasks outside the predefined group to have access to that specific resource (i.e., setup time can be prohibitive and thereby render some amount of resource idle time more cost effective than securing a higher resource utilization rate).
3) Support the premature release of a resource reserved for a group of tasks even if all tasks in the group are not yet complete.

While far from being an exhaustive list of requirements, this list begins to point to the type of flexibility needed. At a minimum, each constraint in the modeling vocabulary is a candidate a relaxation/enforcement control.

One Possible Solution

Modeling Vocabulary

One technology solution, and the approach we have chosen for our work, uses an approach called the "computation of feasible intervals." We have found that this approach to scheduling allows us to model much of the vocabulary mentioned earlier; i.e., we find that we can visualize many of the constraints in the vocabulary as collections of intervals to be intersected with other sets. The re-scheduling problem can be solved quite nicely under this paradigm. It becomes merely the inclusion of a new constraint or two.

Process Control

To satisfy the need for control over the (re) scheduling

process, provide to users a powerful task and resource selection/sorting scheme. This can be done by providing:

- commands that operate not over the entire set of tasks or resources, but a selected subset.
- operations that use two task lists, the previous selection and current one, to create a new, composite selection; e.g., intersect, add, subtract, etc.
- a rich set of selection criteria or "filters"
- varied sort algorithms which operate over the selected list whose sorting effects can aggregate

Another tool for controlling the process is provided through control of the engine's internal switches or "mode." Such switches control how the engine processes various constraints and the direction of the schedule operation.

Finally, users can invoke scheduling/unscheduling commands at will over the selected data. The commands are sensitive to order, so here is where the effects of sorts will be seen. Our experience has been that these controls are sufficient for controlling a variety of scheduling processes. General, simple command sequences can usually provide good results. More advanced users who know their data can implement their own "heuristics" through a customized series of selection, sort and schedule commands. If a macro-recording tool is then given to them, they can record their commands for selecting, sorting, and scheduling for subsequent use in other sessions.

Engine Design

There are many, many design considerations that must be examined when building the engine and it is well beyond the scope of this paper to list many of them, much less discuss them. What will be presented are those considerations we believe are particularly relevant to the (re) scheduling problem itself.

Common Scheduling Function: One choice of the scheduling technology, computation of feasible intervals, requires that all constraints be ultimately represented as a set of intervals. In building an engine using this approach, it would be wise to design the engine's core so that it has no knowledge about how to "interpret" any type of constraint, but instead only "talks" in terms of lists of time intervals. By making such distinctions, we can design an engine core that is independent of the type of constraint being processed. The addition of new constraints then becomes one where a higher level data abstraction is defined with appropriate routines to translate it into intervals. This makes adding new constraints to the vocabulary a reasonably scoped, understood activity.

The inclusion of status data in support of the (re) scheduling process is one that, under this paradigm, influences the translation of a few other constraints as they are communicated to the engine core. In fact, the *only* field that is affected in our work is the task's "duration" field. One immediate benefit can be seen in the basic algorithm that effects scheduling (placement of unscheduled tasks onto the timeline). The same routine is used to perform both scheduling and re-scheduling. The only difference in re-scheduling is in the modification of a task duration's value if it has a non-default status.

Constraint Processing: Tasks have status data that documents state and deltas against original duration values. Scheduling functions always compute an "effective" duration using original and delta information. If no status information has been supplied, the original planned data is also the effective data. If status information has been supplied, effective data will differ from planned data. In either case, the scheduling function is itself unmodified; it needs a duration value and that's what it gets.

Tasks which are completed have an effective remaining duration of zero. Internally, a zero duration task is treated as a milestone with respect to interval calculations. Point solutions have special effects on resource requirements that are reservations -- they are effectively ignored since they cannot affect a resource's availability over an interval of time. Milestones, however, can still effect production and consumption of resources.

Consider a precedence relation between two tasks. If the successor task is completed out of sequence, the engine will "float" it into the future if needed so that the predecessor's constraints are still enforced. The task that floated into the future now is a point solution and therefore its documented resource reservation requirement no longer impacts the schedule.

There are some hidden issues with this approach. First, if the data model has tasks using temporal or relative constraints against the out-of-sequence task it is possible to create a logically inconsistent condition. Secondly, the completion of work out of sequence creates an operational dilemma for successor tasks. The schedule shows successor tasks later in the schedule than the completed but out-of-sequence task. Can these latter tasks be performed? Users can always modify the constraint data to remove these problems, or the engine can be modified to include a more complex behavior. Our experience has been that the current approach is adequate for most needs and the additional complexity required in the engine to effect a more sophisticated behavior warrants a stronger motivation than any we have yet to encounter.

Finally, the tasks data structure can have placeholder field to hold dates of actual task completion if this is important. This data, using the above rationale however, is not used in the scheduling process (though the engine could be modified to use it at the risk of introducing constraint inconsistencies). Should historical data become important, reports can make use of this data.

Status: Tasks have one of three values; waiting, inwork, and completed.

Waiting: this is a task's default state. The documented task duration will be unmodified when passed to the engine. Should the task be delayed, the waiting status may be augmented with a "fence," another instance of an earliest start time constraint. This is needed to support real-world impacts to the schedule such as the delay of a

task so that other tasks can take advantage of an unexpected window of opportunity.

Inwork: Tasks can be continuing or suspended until a user specified date. The suspended feature is needed to support real-world impacts to the schedule. A task which is continuing will be credited with an optionally supplied amount of work that has already been performed and then, during scheduling, placed on the timeline as soon as allowable. If no completed hours are supplied, then the effective duration calculation will not provide any credit.

The optional delta duration is a value to be applied against the documented duration resulting in an effective remaining duration different than the original documented duration. It is used to document a change in the overall duration of the task from the planned value. It can be positive or negative.

Input at the user interface for this data can be quite varied;

- percent complete and, optionally, hours remaining
- hours completed and, optionally, hours remaining
- hours completed since last status operation (if the task is interruptible and worked in many increments) and, optionally, hours remaining.

Completed: Task when complete reduce to milestones on the schedule. Optional values can be supplied that, for reporting purposes, can be used to document how long the task actually took. The delta duration value here describes the change from the documented task duration.

Input at the user interface for this data can be varied but is not as complex as the inwork status. Users can supply total hours worked or total hours worked since last status operation. If no hour data is supplied, use the original duration values.

Interfaces to Legacy Systems:

All data structures and types have a core set of defined functions, including readers and writers. This includes the status data and all commands. Readers and writers use ASCII and should be designed to facilitate users' inspection of data.

One consequence of this approach is that the data/command can be read from and written to a stream. This allows us to easily build interfaces with other systems. A stream can come from an electronic interface with another system or from an opened file. Specifically, status data from other systems can be output to a file in the format of a scheduling command. This file can subsequently be processed by the scheduling engine.

By initially using a file interface scheme, requirements can be partitioned into information requirements and electronic peer to peer communication requirements. Interfacing activities should first ensure that the required information can be supplied to the engine via a file. Once this supplied data can reliably drive the engine's scheduling behavior in an acceptable and expected manner, attention can focus on automated, live electronic data feeds for gathering schedule status data.

Re-scheduling for an operational schedule can make use of the engine's scheduling mode/configuration to obtain desired behaviors. For example, by selecting tasks which have a firm latest completion date, these tasks and their predecessors can be placed onto the timeline first by scheduling backwards from their latest completion dates. Remaining tasks can then be selected and the mode changed to a forward direction. Tasks will be placed onto the timeline around the previously scheduled tasks.

The TimePiece Scheduler

The product we have developed that implements the above approach is called *TimePiece*. Below are some examples of how this product and its predecessor, COMPASS, have been used to solve large, complex (re) scheduling problems.

Aircraft Final Assembly

The schedules controlling the final assembly of a fighter aircraft can be quite large and contain highly inter-related tasks each having a potentially large set of resource constraints. Schedule characteristics for one specific assembly line include:

- 1500+ tasks per aircraft (two distinct precedence diagrams each) with precedence, resource, and relational constraints, group usage, task interruptability, etc..
- Four aircraft in flow at any one time with an expectation to grow to as many as six for a total of well over 9000 tasks represented in the schedule
- (Re) scheduled daily in 54 minutes on a 133 MHz Pentium PC, having 96 MB RAM
- Status data is provided by a legacy, mainframe database to TimePiece via a formatted command file

Avionics Integration Center

A different problem domain using this same technology is one involving an avionics test facility. Many users submit discrete, unrelated requests for time in the facility. Each request can contain some vary complex resource expressions which can document alternate sets of possibilities, any one of which will satisfy the need. Characteristics of one such lab scheduler include:

- Management of approximately 450 discrete resources
- As many as 250 users to submitting requests into the facility
- Support for multiple aircraft programs all of which contend for the same resources
- Daily processing of new requests against the baseline schedule
- Use of e-mail to notify users of reservation times and equipment allocation

This system uses a COMPASS derivative and schedules the facility in less than 10 minutes when running on a multi-user VAX. Efforts have started to upgrade its design

to use the new product, Timepiece.

Use of this facility scheduling system has helped realize: an order of magnitude jump in test hours scheduled (80 hours per week to 700+ per week), and an order of magnitude reduction in time spent scheduling (from 50+ hours per week down to as low as 2 hours per week. It has been in use since the first quarter of 1995 with dramatic results and cited as an aircraft program "best practice."

Scheduling for Space

Our products are also in use to support space related activities. Our older product, COMPASS, is in use to by SpaceHAB, Incorporated, to support preliminary scheduling for its missions. NASA/JSC also uses a COMPASS derivative to schedule one if its facilities, the Systems Engineering Simulator. TimePiece, our latest product, uses the same underlying approach to scheduling and implements lessons learned from these scheduling systems.

Conclusion

At first glance the types of problems encountered in space operations may appear to be sufficiently diverse to require different scheduling approaches. One may be tempted to say that the problem of generating the original schedule and subsequent rescheduling might require different approaches and underlying technology choices. Our experience has been that by understanding the nature of both the scheduling and rescheduling problem, through the choice of a good scheduling technology (calculation of feasible intervals), and through the use of wise design decisions, the same scheduling engine can be used to support a wide range of problem domains, including original schedule generation and its subsequent re-scheduling. What was once a solution requiring a workstation can now be done on a standard PC, and it can be done well.

Reasoning About and Scheduling Linked HST Observations with SPIKE

Laurence A. Kramer, Mark E. Giuliano

Space Telescope Science Institute (STScI)
Baltimore, MD 21218

1. Abstract

Scientific observations executed by the Hubble Space Telescope (HST) are subject to a number of complex and interacting timing and orientation constraints. We classify these constraints as "absolute" and "relative." The former apply to only one observation, whereas the latter apply to two or more linked observations or "visits." Examples of absolute constraints are **"Schedule Visit A BETWEEN Day 10 and Day 20"** or **"Schedule Visit A at an ORIENT in the range 30 to 40 degrees."** Relative constraints express either how the visits should be linked in time (for example **"Schedule Visit2 AFTER Visit1 by 10 days"** or linked in orientation (for example **"Schedule Visit2 ORIENT FROM Visit1 by 20 degrees"**).

This paper will focus on the implementation of relative constraints, how they are combined with absolute constraints, and the problems encountered and solved in combining relative constraints along the orthogonal dimensions of time and orientation.

We also briefly present the SPIKE Plan Window Scheduler (Giuliano 1997), which creates a Long Range Plan by assigning "plan windows" where the visits might feasibly be scheduled, and which optimizes the placement of windows to satisfy a number of criteria.

Since the assigned plan windows are much larger (nominally eight weeks) than the actual times needed to schedule the visits, we confront the problem of "link set flexibility." This implementation allows a great deal of latitude for a short term scheduler to later assign actual spacecraft execution times to visits. We also describe an enhancement of the Plan Window Scheduler to schedule orientation angles along with plan windows for orient linked visits, while confronting computational constraints.

2. Introduction

The SPIKE (Johnston and Miller 1994) software system is used both to determine where Hubble Space Telescope observations are schedulable and to craft a long range observing plan that is flexible while maximizing telescope usage. Scheduling in this large, complex domain is further complicated by dependencies between individual observations, or ***visits***, in an observing program. Approximately 39% of 2,444 HST visits for Cycle 7 are linked by timing and/or orientation constraints, and ensuring that these observations are scheduled correctly and planned optimally is a major operational concern for the Hubble ground system.

In this paper we describe problems that have arisen in reasoning correctly about linked observations and our solutions to these problems. We go on to investigate issues that arise in creating a long range plan that will assure a short-term scheduler as much flexibility as possible to create one-week calendars.

3. HST Domain

Scheduling observations on the Hubble Space Telescope involves first processing a number of interacting constraints to determine where an observation might possibly schedule, and then creating a plan based on where all observations, individually and as a whole, might *best* schedule. In practice, the SPIKE system is used to determine schedulability and a long range plan (***LRP***) over approximately a one-year time period. Other software systems are used to craft a weekly short-term schedule that will actually "fly" on the spacecraft.

Determining those times that are suitable for scheduling depends not only on constraints due to the nature of the Space Telescope and the celestial target being observed, but also constraints that the Principal Investigator (PI) has introduced into the observing program (or ***proposal***) in order to obtain desired scientific goals. For long range planning purposes, the HST and target are responsible for what we call ***absolute constraints,*** constraints that affect only a single observation. The PI, however, may specify conditions that are responsible for both absolute and ***relative constraints,*** constraints relating one observation to another.

3.1. HST Absolute Constraints

Hubble's low Earth orbit, solar panel power requirements, and scientific instrument sensitivities, combined with the target position over time, are responsible for numerous absolute constraints on observation scheduling. First of all, due to instrument sensitivity, the HST must not be pointed within a certain degree range from bright objects such as the Sun and Moon. This will restrict when targets can be viewed to those times when their pointing is not too close to a bright object. In viewing a target, the HST must be oriented both so that it does not overheat, but also to provide adequate power from the solar panels. This ***orientation constraint*** is for most purposes treated as another absolute ***timing constraint***, i.e. those times for which it is legal to achieve a certain orientation. The distinction between orientation and timing constraints, produces another dimension, orthogonal to that of absolute/relative. We will explore these dimensions in some detail below.

In crafting her program, the PI can introduce various absolute constraints on the HST. For instance, it may be desirable to view a target only between certain dates, although the target's window of visibility might actually be much greater. This is expressed as **[VISIT n] BETWEEN <DATE1> AND <DATE2>**. Other constraints might be **BEFORE** or **AFTER** a certain date. Similarly, the PI might want to restrict the spacecraft orientation (roll angle) relative to an instrument aperture to some absolute degree range (roll range), expressed as **[VISIT n] ORIENT <ANGLE1> TO <ANGLE2>**.

3.2. Representation of Constraints in SPIKE

SPIKE implements constraints through the use of piecewise continuous functions or pcfs (Johnston and Miller 1994), which represent schedulability over time. A pcf can be represented as a list of time intervals and ***suitability*** values, ($time_1$ suitability$_1$ $time_2$ suitability$_2$ $time_3$ ...). For the purposes of this paper, however, we will discuss SPIKE constraint representation and manipulation in terms of ***non-zero intervals***, i.e., those time intervals that have non-zero suitability for scheduling, and we will use the notation **(($time_1$ $time_2$) ($time_3$ $time_4$) ...)** to represent the suitable intervals for a given constraint as well as overall schedulability for a visit. Unless there is a need to be more precise, we will use the terms *suitability*, *non-zero intervals*, *constraint windows* and *schedulability* interchangeably.

Each $time_n$ will be denoted by an integral "logical" date, as opposed to a real calendar date. For instance, the constraint that Visit1 be scheduled between November 1 and November 10 or

Visit1 BETWEEN 305 and 314

will be expressed as the non-zero intervals **((305 314))**. {1}

Now suppose suitability for Visit1 due to Sun and Moon avoidance is expressed as the non-zero intervals **((200 312))**. {2}

Then the ***absolute suitability*** for Visit1 is **((305 312))**, {3}

which is derived by intersecting suitable intervals from {1} and {2}.

Similarly, an absolute orientation constraint may be converted to a time suitability function. For instance, the constraint

Visit1 ORIENT 30deg. TO 40deg.

may be equivalent to the non-zero intervals **((302 311))**. {4}

This says that orients in the range 30 to 40 degrees can be achieved between day 302 and day 311. Intersecting {4} with {3}, we get an overall suitability function, or schedulability of **((305 311))**. {Visit1 Suitability}

3.3. HST Relative Constraints

Relative constraints express relationships between two or more visits. PIs have at their disposal a rich language for expressing both timing and orientation constraints between distinct observations. Continuing our previous example, suppose there is a second observation, Visit2, which should be scheduled after Visit1. Visit2 has combined schedulability due to absolute constraints of **((308 314))**. {Visit2 Absolute Suitability}

In addition the PI desires the following constraints:

Visit2 AFTER Visit1 by 5 to 10 Days {Rel. Constraint 1}
Visit2 SAME ORIENT AS Visit1 {Rel. Constraint 2}

The former expresses a ***relative timing constraint***, and the latter a ***relative orientation constraint***.

4. Combining Absolute Constraints and Relative Constraints

In order to derive the suitability for visits linked by relative constraints, we express as non-zero intervals the effects of each relative constraint on all the visits linked by that constraint, and intersect the relative suitability intervals with the absolute constraint suitability intervals for each visit. Considering first Relative Constraint 1 above, since Visit1 is schedulable between 305 and 311, Visit2 may be scheduled between 310 and 321 (*at least* five days after the earliest date and *at most* ten days after the latest date). Thus, Visit2's ***relative suitability function*** is

((310 321)). {Visit2 Relative Suitability}

Similarly Visit1 may be scheduled *at most* ten days earlier than Visit2's earliest date and *at least* five days before

Visit2's latest date, producing a relative suitability for Visit1 of

((298 309)). {Visit1 Relative Suitability}

Now relative suitability intervals can be combined directly with the absolute suitabilities to produce suitability functions for Visit1 of

((305 309)) {Revised Visit1 Suitability}

and for Visit2

((310 314)). {Revised Visit2 Suitability}

In the general case, there may be more than two visits linked by a relative constraint, and a number of relative constraints which interact with each other. This could necessitate several iterations of constraint propagation until suitability functions of the linked visits stabilize to a final value.

It is important to note that once relative constraints are introduced between visits, the strength of what the suitability function expresses for a visit diminishes. For all visits not linked by relative constraints we can state that they will be schedulable at *every* time with non-zero suitability in their suitability function. For visits linked by relative constraints we can only state that a suitability function represents *potential* schedulability, and once a visit in the link set becomes scheduled there *will exist* time intervals with non-zero suitability at which other visits in the link set will be schedulable.

This is easy to see from the example above. Initially, 310 is a perfectly legal date to schedule Visit2, but once Visit1 is scheduled on 306, the interval up to 311 becomes unsuitable for Visit2 (due to the After constraint). In SPIKE we handle this problem by introducing "***execution time constraints***" after a visit is placed on a short term schedule, which have the effect of collapsing a visit's suitability function to its actually scheduled time.

4.1. Propagating Relative Suitability

We state a property of *potential schedulability* to which any algorithm for propagating relative suitability must adhere:

Property of Potential Schedulability

Define a ***link set***, L, as the transitive closure of all visits in a proposal, P, mutually reachable through relative constraints (both timing and orient). Visits in a proposal that are not reachable through relative constraints form singleton link sets.

Then, For every visit V in L and for every point in time where V has non-zero suitability, there must exist a point in time having non-zero suitability for every visit linked to V, these time points representing a fully feasible schedule for L. In other words, suitability functions for a link set must contain no *a priori* unschedulable time intervals.

The method for propagating relative suitability for visits in a proposal can be expressed algorithmically as follows:

Algorithm for Propagation of Relative Suitability

```
For each visit V
    set suitability(V) = absolute-suitability(V)
End for
set Changed = True
While (Changed)
    Changed = False
    For each visit V
        set S = suitability(V)
        For each rel. constraint C which affects V
            set R = Apply C to V
            set S = Intersection (S, R)
        End for
        If S <> suitability(V)
            set suitability(V) = S
            set Changed = True
    End for
End while
```

4.2. Propagating Relative Orient Constraints

In the discussion above we have glossed over how relative *orient* constraints are propagated. As with absolute orient constraints, a good approach might be to somehow convert the relative orient constraint roll range to a relative timing suitability function and then propagate this suitability as we have described.

Consider Relative Constraint 2. It states that Visits 1 and 2 must be scheduled at the same orientation. In other words, if Visit1 is scheduled at 59 degrees, Visit2 must be scheduled at 59 degrees. If Visit2 is scheduled at 120 degrees, so must Visit1 (there is no preferred "first" visit to an orientation constraint).

To compute a relative suitability function for this constraint for Visit2, recall that Visit1 is schedulable between days 305 and 309. Assume that over this time period orientations in the roll range from 34 to 38 degrees are allowed. We then consider during which time periods Visit2 can achieve a roll in the [34 38] range, say days 308 to 312. Then, Visit2's initial relative suitability function due to Relative Constraint 2 and Visit1 is

((308 312)). {Visit2 Relative Orient Suitability}

This suitability will be combined with Visit2's absolute suitability to produce a new suitability. A relative suitability function for Visit1 based on roll ranges that Visit2 can achieve is similarly computed. Suitabilities continue to be propagated in this fashion until they become stable (they no longer shrink or become zero).

4.3. A Problem Reasoning About Relative Orient Constraints

SPIKE was originally designed to use this method (Sponsler 1990) of propagating relative orient constraints, but problems with converting back and forth between roll ranges and time suitability functions were uncovered and a different propagation methodology has been developed.

In order to illustrate the problem, we extend the preceding example. We now have three visits in the link set with several constraints:

Visit1 ORIENT 30deg. TO 40deg. {Absolute Constraint}
Visit2 SAME ORIENT AS Visit1 {Rel. Constraint 1}
Visit3 ORIENT 10deg. FROM Visit2 {Rel. Constraint 2}

Suppose all the absolute constraints on Visit1 imply a schedulability period of day 302 to day 311 for that visit. Converting this to an orient range and propagating to Visit2, we compute that Visit2 is schedulable between day 308 and day 316. Now, to propagate constraints to Visit3, we must convert Visit2's suitable time interval to a roll range and then apply an "orient-from" operator to this roll range, finally converting it to a time interval for Visit3. Suppose the available roll range for Visit2 in the day 308 to day 316 time frame is [30 41]. Then Visit3 must be scheduled 10 degrees from this, or in the [40 51] range. We convert this to a time interval for Visit3, say day 340 to day 350.

While the schedulability intervals for Visits 1 and 2 can be guaranteed to be good, it is quite possible that the interval for Visit3 may contain false positives for scheduling opportunities. By this we mean that even before any visits in the link set are scheduled there may exist illegal time intervals in Visit3's suitability function.

Suppose both Visit1 and Visit2 are scheduled somewhere within their legal time intervals and the short term scheduling engine attempts to schedule Visit3 on day 350, but finds that it can only schedule in the roll range [51 55]. To schedule anywhere in that range will violate the orient constraint of 10 degrees from Visit2 (whose range is [30 40]), and thus the Property of Potential Schedulability (day 350 is a priori unschedulable). How is this possible?

The error has cropped up when Visit2's suitable time frame is converted to the roll range of [30 41]. In doing so, information has been lost about the absolute orient constraint on Visit1 of [30 40], which Visit2 must implicitly honor through the same orient constraint. Note that this is not an error in computation, but a true flaw in our methodology.

While it is true that the time frame day 308 to day 316 for Visit2 is the full extent at which a [30 40] roll range can be achieved, it may be *possible* to achieve a roll angle of 41 during that interval, as well. A sound algorithm should exclude 41 from the roll range of Visit2 and 51 from the roll range of Visit3.

4.4. A Better Algorithm for Propagating Relative Orient Constraints

Our solution to this problem is that while propagating relative orient constraints, roll range information must be preserved throughout the propagation, and not converted to final time suitabilities until the propagation has stabilized. To implement this we revise the Algorithm for Propagation of Relative Suitability roughly as follows:

Revised Algorithm for Relative Suitability Propagation

Define a Total Roll Restriction RR for a Visit V to be the set of roll angles from [0 360) at which it is feasible to schedule V.

```
For each visit V
    set suitability(V) = absolute-suitability(V)
    set RR(V) = [0 360)
End for
set Changed = True
While (Changed)
    Changed = False
    For each visit V
        set S = suitability(V)
        set Roll = RR(V)
        For each rel. constraint C which affects V
            set values R, RR_r = Apply C to V
            set S = Intersection (S, R)
            set Roll = Intersection (Roll, RR_r)
        End for
        If S <> suitability(V) or Roll <> RR(V)
            set RR(V) = Roll.
            set suitability(V) = S
            set Changed = True
    End for
End while
```

Roll restrictions that are intersected to produce a total roll restriction on a visit include restrictions due to legal roll angles for the visit due to its absolute suitability, restrictions due to absolute orient constraints, the actual angle at which a visit is scheduled (analogous to an execution time constraint), affects from other linked visits through relative orientation constraints, etc.

Reworking our prior example, we illustrate this new algorithm: Suppose the absolute timing constraints on Visit1 imply a schedulability period of day 302 to day 311 for that visit and a possible roll restriction of [23 42]. We calculate a preliminary total roll restriction for Visit1 of [30 40] by intersecting with the absolute orient constraint roll range. Propagating to Visit2, its roll restriction must also be [30 40], by the Same Orient constraint. Assume that this is feasible, otherwise Visit1's roll restriction would be further constrained. Based on this roll restriction, Visit2 is schedulable between day 308 and day 316. To propagate constraints to Visit3 apply an "orient-from" operator to Vist2's roll restriction, producing a roll restriction of [40 50] for Visit3. We convert this to a time interval for Visit3, day 340 to day 349.

Notice that the end result of this process drops the problematic day 350 from Visit3's suitability function and ensures that all visits will be scheduled where both their timing and orient constraints will be strictly obeyed.

4.5. Yet Another A Problem Reasoning About Relative Orient Constraints

Implementation of the improved algorithm for propagating relative orient constraints has greatly reduced the incidence of false positives in computing visit schedulability. However, it has recently come to our attention that this solution is not completely correct. Consider the following simple example:

Visit2 AFTER Visit1 by 80 to 90 Days {Rel. Constraint 1}
Visit2 ORIENT 180deg. From Visit1 {Rel. Constraint 2}

Suppose there are no other constraints on either visit and the planning interval we are considering is for a full year. Both Visits initially have unlimited schedulability, which is only slightly constrained after considering the After constraint. Visit1 will have a suitability function of

((1 285)), {Visit1 Suitability}

while Visit2's suitability will be truncated at the other end:

((81 365)). {Visit2 Suitability}

Assume that due to these large time intervals neither Visit1 nor Visit2's roll range is restricted beyond the full [0 360) range. Applying any Orient From offset to a full roll range returns a full (unrestricted) roll restriction of [0 360). Therefore both visits' suitability functions remain unchanged due to the Orient From constraint.

Unfortunately this solution is not correct. It turns out that if Visit1 is scheduled on day 1, Visit2 is not schedulable on day 81 as we might expect. This is due to the fact that for most celestial targets, HST's legal nominal roll range only varies by about a degree a day, and even allowing for as much as 30 degrees off-nominal, it would be impossible to span 180 degrees (required by the Orient From constraint) in 80 days.

Again, the Property of Potential Schedulability has been violated, as our suitability functions contain days that have no schedulability. What's worse, in this case *all* times in the suitability functions are unsuitable as it is impossible (at least for this example) to schedule a visit 80 to 90 days after another and 180 degrees from it!

How could such an egregious error be missed? In practice, such underconstrained proposals where the orient and timing links line up so perversely are very rarely encountered. We designed our new algorithm specifically to handle orient information, though, so what is its flaw?

The error occurs in propagating roll ranges as global entities which apply to a visit without regard to the time interval. In actuality a roll range for a visit is only "good" for a restricted period of time. For instance Visit1 may have an allowable roll range of [20 50] on day 1, an allowable range of [21 51] on day 2, and so on. By day 10 the roll range might be [29 59]. In the time interval ((1 10)) the allowed range applicable over the entire interval would then be [29 50].

This example, although typical, is very arbitrary. For some targets the roll range will remain relatively constant over time, and then change radically in a one-day interval. In theory then, it is virtually impossible to craft a combined relative timing and orient link propagator that will not violate the Property of Potential Schedulability.

4.6. A (Best Effort) Solution To The Orient Propagation Problem

In practice though, we can come arbitrarily close to a sound propagation algorithm by modifying our current algorithm to build up a suitability function in small time increments. In other words, we run the same algorithm but one day (or one hour) at a time and aggregate a suitability function from these iterations. Clearly there is a time slice at which this becomes *computationally* infeasible, but initial prototypes have shown that a one-day sampling period should be both tractable and correct for almost all real world HST observation programs.

5. The SPIKE Plan Window Concept

We have gone into some detail presenting problems and solutions for reasoning with linked observations in an HST observing program. Now, we briefly present the SPIKE Plan Window concept and the SPIKE Plan Window Scheduler, which generates a Long Range Plan (LRP) for a

cycle's (typically one to two years) worth of proposals. We then go on to confront the issue of "link set flexibility."

In constructing a Long Range Plan we encounter the conflicting goals of producing a plan that should be as stable as possible over time, while allowing frequent revision of proposals necessitating changes in where they can schedule (See (Giuliano, 1997) for a full discussion of these issues). Historically, the SPIKE system generated an LRP where visits were assigned to one-week windows, allowing a short-term scheduler to assign an actual time within the week.

This method proved to be somewhat inflexible as replanning took place over time, often ending up with weeks where there were too many visits to schedule, and other weeks that were undersubscribed. If a visit missed its one-week window, often it could only be rescheduled a full year later.

To address these problems we implemented the SPIKE Plan Window Scheduler. Instead of scheduling visits to a fixed week in the plan, it schedules them to an eight-week "***plan window***," any week in which is suitable for scheduling (of course, many visits are so constrained as to have constraint windows less than eight weeks in duration). This implementation has proved in practice to lead to a more stable while flexible LRP. Each week the short-term scheduler has a pool of visits from which to select and craft an efficient schedule. If a visit cannot be placed on the current calendar, it can usually be placed on a later calendar within its assigned plan window.

5.1. The SPIKE Plan Window Scheduler

The SPIKE Plan Window Scheduler works roughly as follows:

Plan Window Scheduler Algorithm

Given an input LRP (null for the first iteration) and a set of link sets (including singletons) to schedule, Do for each link set L:

1. Iterate (one day at a time) over time for the entire planning period and the link set's constraint windows, assigning plan windows to each visit in the link set.
2. Score each set of plan window assignments based on various scheduling, planning, and resource criteria.
3. Make a final assignment of plan windows, those with the highest score, to L.

If there are any oversubscribed regions in the plan so generated:

1. Execute a stochastic repair algorithm which selects link sets to reschedule.
2. Attempt to reschedule these link sets subject to resource and other criteria.

Save the resulting plan as the new LRP.

In practice a new LRP is generated on a daily basis. Generally, if a link set has been assigned plan windows in today's LRP, and no changes are made to that link set, it will retain its original plan windows in tomorrow's and succeeding LRPs.

5.2. Assigning Plan Windows *and* Angles

We have described in general how SPIKE assigns plan windows for link sets, but have neglected the subject of assigning orientation angles. For visits that are unaffected by orientation constraints, an orientation is typically assigned (outside of SPIKE) at the nominal orientation. For orient linked visits, though, this has up until very recently been a tedious and error prone manual process.

An enhancement to the SPIKE Plan Window Scheduler has been to schedule orientation angles for orient link sets. We discuss our implementation and some time complexity challenges we have faced.

Recall that as an output of the Revised Algorithm for Relative Suitability Propagation we compute a Total Roll Restriction RR for each orient linked Visit V. To schedule "optimal" orientation angles and plan windows we implement the following:

Algorithm for Assigning Plan Windows and Orient Angles

For a link set L having relative orient links Define $Visit_{first}$ to be that visit in the link set with the most highly constrained roll restriction, RR. Call this RR_{first}.

1. Select an angle, A, from RR_{first} and set RR_{first} = [A A]. Propagate this restriction through the link set, constraining each linked visit.
2. For each $Visit_i$ orient linked to $Visit_{first}$, select and propagate an angle from its constrained RR_i.
3. Execute the Plan Window Scheduler Algorithm as previously outlined (iterate through time, testing plan windows, assigning the most highly score plan windows, *and in addition*, orient angles).
4. Until all valid combinations of angles from (RR_{first}, RR_i) have been exhausted, return to step (1) and repeat.

Note that this algorithm can be quite time consuming. Basically, to schedule each link set L with relative orient links takes N * T_L, where N is the number of valid combinations of angles for L and T_L is the time it would have taken just to assign plan windows to L.

What is a bound for N? The worst case value for N can be computed as follows:

Let m be the number of orient linked visits in L, then an upper bound on N is 360^m (assuming sampling at 1-degree increments).

In practice N is typically much smaller than this value, both because each RR_i is generally much smaller than [0 360), and also because the RR_i are not independent of each other. For example, suppose we have a link set L with m visits linked by a Same Orient constraint, then we can bound N as 360, no matter how great m is. In this case, selecting an angle from RR_{first} constrains each angle in RR_i to be identical, thus limiting the number of distinct combinations to be 360.

Since most link sets happen to be simple Same Orient link sets, and since most roll ranges are far more constrained than a full 360-degree range, a conservative average case time estimate for N is 180. In other words, for link sets with orient links, assigning plan windows *and* angles takes about 180 time longer than assigning plan windows alone.

As we previously mentioned, the Plan Window Scheduler is run nightly to generate a new LRP, so the enhancement of scheduling angles and plan windows must not take so long as to cause the LRP run to be longer than about six hours.

In order to reduce the time necessary for selecting angles, we have introduced a sampling algorithm, which significantly reduces the average case time for scheduling plan windows and angles:

Grid Search Algorithm for Sampling Angles

1. For each $Visit_i$ in L, set the grid size GS_i = 5 * (ceiling (size (RR_i) / 90)). I.e., the grid size increases by five, for each 90 degrees of roll range.

2. Execute the Algorithm for Assigning Plan Windows and Orient Angles, sampling angles at intervals of GS_i for each $Visit_i$. After the "grid" has been fully sampled, select the two highest scored angles, and search in both directions in integral increments from these angles, terminating the search when half the grid size or a worse angle has been reached for each angle and each direction.

Given our conservative average case scenario of RR_i = 180, GS_i will be equal to 10, and thus the number of samples will be approximately 18. Thus we have achieved an order of magnitude speed up. Preliminary tests with sampling angles shows that the domain is smooth enough that we sacrifice no accuracy by sampling compared to the brute force approach, and miss no local maxima.

5.3. Link Set Flexibility

When SPIKE creates plan windows for linked visits it needs to ensure that the windows allow flexible scheduling of all visits in the set. In general if a visit has at least eight contiguous weeks of suitability, it makes sense to assign it an eight week plan window, thus maximizing scheduling opportunities. We have uncovered a counterintuitive result, however, that extending the plan window for one visit in a link set may actually greatly *decrease* scheduling flexibility for other visits in the link set. Consider the following example:

Visit1 is suitable days 1 to 30. Visit2 is suitable days 21 to 50. In addition we have the constraint

Visit2 AFTER Visit1 by 20 to 30 Days.

If Visit1 is scheduled on day 1 then Visit2 is schedulable days 21 to 31. In this case the full link tolerance is available. In contrast if Visit1 is scheduled on day 30 then Visit2 is only schedulable on day 50. In this case only 10% of the link tolerance is available.

SPIKE should not include day 30 in the plan window for Visit1. In the above example creating a window for Visit1 from 1 to 20 will ensure that the minimum size window for Visit2 is 10 days long.

Our solution to the flexibility problem is that SPIKE should create plan windows which maximize the guaranteed minimum window size of the entire link set. A technical definition of the concept is developed below.

The discussion given is independent of the type of link set (e.g. timing, or relative orient). The algorithm measures flexibility in terms of days, which is sufficient for a Long Range Plan. However, the algorithm could be modified to measure flexibility in finer units if desired. In general SPIKE chooses plan windows which optimize a set of criteria out of which link set flexibility is one criterion. In practice, then, flexibility may be somewhat compromised to benefit other planning and scheduling criteria.

Algorithm For Ensuring Link Set Flexibility

Let L be a link set (i.e. the transitive closure of timing and relative orient links in a proposal).

Let Pw be a set of plan windows for the link set L, V is a visit not equal to the first visit in L, and D is the day where the first visit of the link set is scheduled.

Then define Actual(L,V,D) as the raw number of days that V can be scheduled.

Define Actual_Flex(L,D) = Min for all V in L {Actual(L,V,D)}

Define First(L,Pw) to be the window for the first visit in the link set.

Define GMWS(L,Pw) as the guaranteed maximum window size for link set L given that it has plan windows Pw.

We can now give a formalism for computing the guaranteed maximum window size:

GMWS(L,Pw) = Min (size(First(L,Pw)), Min for all D in First(L,Pw) of { Actual_Flex(L,D) })

The GMWS measure can be used as an evaluation criterion for selecting plan windows. In a world with no computation costs we would generate all possible plan windows for a link set and then evaluate them subject to this criterion. However, we cannot possibly generate all possible combinations of windows.

A practical approach is to prune times from the plan window for the first visit in a link set which do not maximize GMWS. Given a candidate set of plan windows for a visit determine the subset of the assignment for the first window in the visit which maximizes the guaranteed minimum size window for all the visits in the link set.

Define Rest(L,Pw) to be the plan windows for the visits other than the first visit in the link set L.

Prune(L,Pw) = Determine the subset S of First(L,Pw) which maximizes { GMWS(L, S union Rest(L,Pw) }

The scheduler would use the flexibility code in two ways:

1. Given a candidate window starting in a day optimize the flexibility using the prune operator.

2. Use the GMWS measure as a criterion to compare different plan windows.

Two additional issues need to be addressed.

1. Below a certain link tolerance SPIKE does not have the constraint accuracy to measure flexibility. For example, SPIKE could not meaningfully measure flexibility for the link **Visit2 After Visit1 by 50 days plus or minus 12 hours.**

2. For chain links we may want to repeat the flexibility procedure when the first visit becomes executed. After the first visit becomes executed the second visit becomes the new "first" visit.

6. Summary

Over the past seven years the SPIKE system has been used operationally in the planning and scheduling process for the Hubble Space Telescope. In refining the system, we have tackled thorny issues related to reasoning about and scheduling linked observations. Recent advances include a better approach to combining constraints along the orthogonal dimensions of time and orientation, and creating a more flexible Long Range Plan.

Acknowledgment

We would like to thank Wayne M. Kinzel of STScI for his contributions to and clarifying discussions of this work.

References

Giuliano, Mark 1997. *Achieving Stable Observing Schedules in an Unstable World.* ADASS '97, Sonthofen, Germany.

Johnston M.; Miller G. 1994. *Spike: Intelligent Scheduling of Hubble Space Telescope Observations.* in Zweben M., and Fox M. eds. *Intelligent Scheduling* (San Francisco: Morgan-Kaufmann), ISBN 1-55860-260-7 (1994), pp 391-422.

Sponsler, J.L. 1990. *The ORIENTATION Constraints.* SPIKE Technical Report Number 1989-21, Revision B, STScI.

Hubble Space Telescope Planning and Scheduling Problems

Jack Leibee
HST Operations Manager
Goddard Space Flight Center, Code 441
Greenbelt, Maryland 20771
jleibee@mail.hst.nasa.gov

Abstract

The Hubble Space Telescope (HST) is a highly complex, Low Earth Orbiter (LEO) spacecraft. Extensive and time consuming processes are required to schedule and execute scientific observations due to occultations, passage through the South Atlantic Anomaly, viewing restrictions in terms of the sun and moon, and availability of guide stars. Because of the hardware design, momentum management and slewing are constrained and have to be carefully managed; the scientific instruments provide limited flexibility in terms of operations; and the on-board data recording and playback systems limitations prevent full utilization of space to ground communications, decreasing observational efficiency. These and other constraints have forced the HST operation of HST to be mostly pre-planned, with limited real-time commanding. Attempts to provide "branching" - originally a Level I requirement - in which one of several pre-planned observations would be selected in real-time, proved impossible to implement.

Although significant progress has been made since launch, some planning and scheduling processes are difficult to optimize or even improve. The HST Project is addressing some of these problems as part of VISION 2000, a program to reengineer the HST ground system and flight computer. Changes to both the planning and scheduling system and the command and control center are being made in order to simplify the operations of HST and reduce operational costs.

Aside from the technical problems presented, some planning and scheduling complexities trace back to decisions made well before the launch of HST. At Goddard Space Flight Center (GSFC), planning and scheduling system development and operations for spacecraft (including command load generation) are typically performed by a single organization. This provides a single user interface. But for programmatic reasons, it was decided that for HST the "science" aspects of planning and scheduling would be performed by the Space Telescope Science Institute (ST ScI) and the "mission" aspects, along with command load generation, would be performed at GSFC. The mission scheduling process is driven by the science planning inputs. As a result of this split responsibility, two major problems occurred. First, there is a duplication of functionality with the same functions being implemented differently. For example, since occultation computations for the Fine Guidance Sensors were computed differently, the mission planning system would sometimes reject science schedule inputs and the science schedule would require regeneration. Second, outputs from mission scheduling were required as inputs for the science scheduling. For example, the science scheduling system needs knowledge of the space to ground communications resources and relies on the mission scheduling system to secure these resources. However, the resources can't be requested until the science schedule is generated. Multiple iterations between science scheduling and mission scheduling are required to resolve this type of problem.

Major improvements are underway. The mission scheduling function has been moved to the ST ScI and put under the control of a single organization. It will eventually be integrated with the science scheduling system which will result in major improvements: elimination of redundant modeling; earlier constraint checking and speedier replanning in response to missed observations or spacecraft anomalies. Due in part to the integration of the two systems as well as to changes in the science planning and scheduling process, observing efficiency has more than doubled since launch.

The HST command and control center plays an integral role in the planning and scheduling process since it makes any necessary changes to space to ground communications schedules during the execution of the observing program. Although the Tracking and Data Relay Satellite System (TDRSS) resources are scheduled at least one week in advance, numerous changes are required due to such factors as: shuttle launches, equipment outages, and unexpected new communication needs (e.g. support other spacecraft emergencies). The goal is to automate this process in VISION 2000. Specific TDRSS resource schedule requests are made 14 days in advance for a one week period. Because the user of TDRSS doesn't have *a priori* knowledge of TDRSS availability, scheduling is

done "in the blind". It isn't until the rejected schedules are returned and services negotiated that a final schedule can be constructed. To expedite this process, "generic" scheduling or developing a highly reliable real-time scheduler are under study with the GSFC's Network Control Center (NCC).

Another complexity is integrating new HST Scientific Instruments (SIs) requirements into the planning and scheduling system. System upgrades to accommodate their enhanced performance has been an expensive and extensive process which needs to be simplified. The solution of retrofitting a new SI to the existing system is unacceptable since much of that SI's scientific capability would be unrealized.

A Planning, Scheduling and Control Architecture for Advanced Life Support Systems

V. Jorge Leon
Texas A&M University
College Station, TX 77843-3367
leon@entc.tamu.edu

David Kortenkamp and **Debra Schreckenghost**
Metrica Inc., Texas Robotics and Automation Center
NASA Johnson Space Center – ER2
Houston, TX 77058

Abstract

This paper describes an integrated planning, scheduling and control architecture for robotics and advanced life support systems. The distinctive characteristics of controlled ecologies and the requirements for planning, scheduling and control architectures are presented. Next, the main components of the proposed architecture are described, and the interaction among the user, the intelligent planner, the generic scheduler, and the crop planner and scheduler is illustrated with a hypothetical scenario. Some successful implementations of components of the architecture and current efforts are also mentioned.

Introduction

We are developing a completely integrated planning, scheduling and control architecture for robotic and life support systems. The operation of a Controlled Ecological Life Support System (CELSS), either aboard a space station or ship or on the surface of the Moon or Mars, will require an intelligent monitoring and control system that can react quickly to short-term environmental changes while planning and scheduling for long-term effects of current actions. A CELSS must sustain a moderate size crew for a number of years with minimum re-supply of mass. Distinctive of this type of environment is the active participation of biological agents (e.g., humans and plants) in a system that possesses mass-closure. From the perspective of integrated planning, scheduling and control, important characteristics of advanced life support systems are:

- The simultaneous presence of difficult constraints including conservation / regeneration of mass, crew availability, space availability, and energy limitations (Leon 1995).
- Non-linear system behavior and long-term dynamics together with agents that respond indirectly to control signals (Auslander 1981; Colombano 1981; MacElroy 1981).
- The requirement of rapid response to environmental changes that pose danger to the crew, with consideration to the effect on long-term system stability of the corrective actions.
- The need to automate labor intensive tasks to offload the crew from time consuming and hazardous working conditions.

Each of these four areas is mission-critical, in the sense that an intelligent architecture must deal with all of them to ensure success. Dealing with the first problem requires reasoning about time and other resources, and scheduling those resources to avoid conflicts while managing dynamic changes in resource availability (e.g., decreasing food stores). Dealing with the second problem requires the ability to plan for a set of distant goals and adapt the plan, on-the-fly, to new conditions. Dealing with the third problem requires tight sense-act loops that maintain system integrity in the face of environmental changes. Finally, dealing with the fourth problem requires an integration of robotic control and scheduling with the overall monitoring and control of the life support system.

Requirements for an integrated planning, scheduling and control architecture

The effectiveness of an integrated planning, scheduling and control architecture for CELSS will depend on to what degree the following requirements are satisfied:

- Interactive planning and scheduling. Given the uncertainties associated with long-missions it would be impossible to automatically generate a plan, pass it to the scheduler, and be done with it in one iteration. The architecture must support the ability of move easily between planning and scheduling in an iterative process, allowing the user to provide significant feedback, if desired.

- User-definable abstraction levels for planning and scheduling. The decisions under consideration may be short-term in the order of minutes, to long-term in the order of years. The architecture must be able to move between various time and granularity scales presenting the correct level of information to the user at different decision situations.

- Flexibility. The user should be able to revise implemented solutions, as well as generate new solutions. The architecture should allow the user to play "what-if" games, evaluating different hypothetical scenarios while, at the same time, the architecture is executing the current scenario.

- The architecture must present a common view of the system to the user. Even though the architecture may be built of many different modules, the user's view should be centered on the kinds of functions the user desires, the problems they want to solve, and the kinds of information the want to see.

- The architecture must explicitly recognize multiple performance criteria. The user must be presented with information that depict the dependencies and compromises among these often conflicting criteria.

- The architecture must be open to facilitate its integration with other systems.

Our motivation is to produce a architecture that satisfies the above requirements. The system will allow the crew to build plans and schedules, play "what-if" games with those plans and schedules, then have the architecture execute them while presenting a consistent view of the state of the system to the use. Such an architecture will greatly increase the effectiveness of the crew in CELSS environments and greatly increase the scientific results of CELSS experiments.

An Integrated Architecture

The integrated architecture is based on the several lines of on-going research at NASA, Metrica and Texas A&M University. Figure 1 shows the complete architecture. The important components of the architecture are:

- A multi-tiered intelligent control system called 3T. The control system combines a reactive tier with a deliberative planning tier, both mediated by a middle tier conditional sequencer. This allows for long-range planning to take place while, at the same time, the system can react to immediate environmental events. This system is being used in several on-going NASA JSC projects (Bonasso *et al.* 1995).

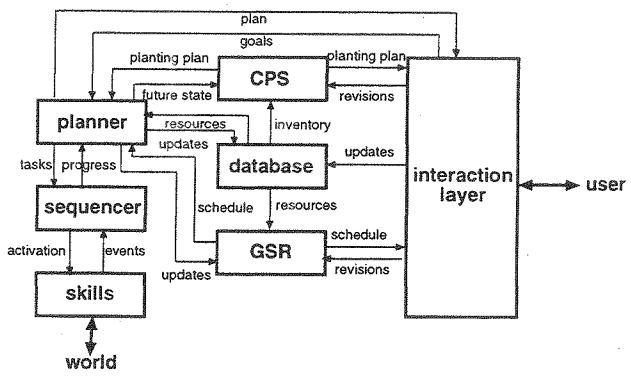


Figure 1: The complete planning, scheduling and control architecture for advanced life support systems.

- A scheduling methodology called the Generalized Scheduler/Rescheduler (GSR) that allows for the use of static, off-line schedules in uncertain environments (Leon & Balakrishnan 1995).

- A crop scheduling and planning methodology that can be used by the planner to decide what crops to plant at what time (Leon 1995; 1996).

- An interaction layer that mediates user interaction with the entire system. The interaction layer hides the various components of the architecture from the user, allowing him or her to concentrate on achieving tasks (Kortenkamp *et al.* 1997).

The rest of this paper will concentrate on the integration of the 3T control system with GSR, as this is the most mature part of the architecture. Brief mention of the other architectural components will be given, followed by an example scenario.

3T architecture

The 3T architecture separates the general intelligent control problem into three interacting layers or tiers:

- A set of hardware-specific situated skills that represent the architecture's connection with the world. The term "situated skills" is intended to denote a capability that will achieve or maintain a particular state in the world.

- A sequencing capability that can activate the situated skills in order to direct changes in the state of the world and accomplish specific tasks. This tier of the architecture is implemented using Reactive Action Packages (RAPs) (Firby 1989).

- A deliberative planning capability which reasons in depth about goals, resources and timing constraints.

We are using a state-based non-linear hierarchical planner known as AP (Elsaesser & MacMillan 1991) for this portion of the architecture.

The architecture works as follows: the deliberative layer takes a high-level goal and synthesizes it into a partially ordered list of operators. Each of these operators corresponds to one or more RAPs in the sequencing layer. The planner places these RAPs on the RAP agenda. The RAP interpreter (sequencing layer) decomposes each RAP on the agenda into other RAPs and finally activates a specific set of skills in the reactive layer. A set of event monitors is also activated to detect certain world conditions and notify the sequencing layer. The activated skills will move the state of the world in a direction that should cause the desired events. The sequencing layer will terminate the skills, or replace them with new skills, when the monitoring events are triggered, when a timeout occurs, or when a new message is received from the deliberative layer indicating a change of plan.

Related work The integration of intelligent planning and control is not a new topic in the artificial intelligence research community. The Cooperative Intelligent Real-time Control Architecture (CIRCA) (Musliner, Durfee, & Shin 1993; 1995) is designed to support both hard real-time response guarantees and unrestricted AI methods that can guide those real-time responses. CIRCA has an AI subsystem (AIS) reasons about high-level problems that require its powerful but uncertain reasoning methods, while a separate real-time subsystem (RTS) uses its predictable performance characteristics to deal with low-level problems that require guaranteed response times. For a good overview of real-time AI work see (Musliner *et al.* 1995).

Simmons' Task Control Architecture (TCA) (Simmons 1990) has been successfully used on a number of real-world robots, but it is very different from our architecture. There are essentially no tiers in TCA. A task net is constructed for the robot which is similar to a task-net in RAPs. Each node in the task tree can be decomposed further or can be a primitive which interfaces with the robot, or other nodes, through a sophisticated message-passing algorithm. These messages are processed through a central router, and thus TCA is more like a robot operating system. There are no explicit representations for expressing relationships among tasks. TCA task trees are manipulated directly by C function calls. Therefore, it is incumbent on the programmer to mentally compile the desired control constructs into the appropriate calls.

The Guardian architecture of Hayes-Roth (Hayes-Roth 1995) is a blackboard architecture designed for controlling embedded (though not necessarily embodied) agents. The architecture is divided into a cognitive component and a perception/action component. The perception/action component is controlled by the cognitive component. Thus, the Guardian architecture is similar to ours, but with sequencing and deliberation performed by the same mechanism. The deliberative component can modulate the performance of the perception/ action component, as well as its own performance, according to the current situation in the world. Guardian embraces traditional AI representation, but does not commit to any particular representation for describing the interrelationships among tasks.

The 3T architecture shares many aspects of Cypress (Wilkins *et al.* 1995). Our AP planner has similar expressive power at an abstract level as SIPE; RAPs compares favorably with PRS. But because RAPs were designed to allow integration with conventional AI planners, we did not have to write an interlingua such as ACTs to achieve such integration. Additionally, Cypress does not specify a canonical interface to the control tier as does 3T.

Generic Scheduler / Rescheduler (GSR)

Given a world's current state-of-affairs, planning specifies a sequence of tasks and required resources to achieve a given goal. Scheduling refers to the time ordering of activities given in plan such that a given objective function is optimized. This objective function may comprise multiple performance evaluation criteria. The capacity of each resource is explicitly considered, as well as, any other conditions required by the plan. Rescheduling refers to the "repair" of a given schedule. The "repair" can be triggered (by the user or automatically) by the occurrence of disrupting events, or when "sufficient" new information about the state-of-affairs becomes available. Rescheduling must minimize the impact that the proposed changes have on system performance. Also note that rescheduling assumes the existence of a schedule.

GSR incorporates multiple-objectives and implements user-interactive search for efficient solutions. The input to GSR is the plan generated by the deliberative layer of 3T. The plan contains information about the required tasks, potential resources required, time constraints, and precedence relations among activities. The output of GSR is a detailed schedule with exact start and finish times for each task and the corresponding resource assignments.

The scheduling engine is based on problem-space based neighborhood generation and search [19]. This approach is computationally efficient, produces good-quality solutions, it is easy to implement, and is very

flexible allowing the incorporation of most operating conditions in the scheduling model. By carefully manipulating the attribute-weights and other parameters of the heuristic, and the weights of the criteria in the objective function, the user can "direct" the search to regions of the solution space that contain satisfactory schedules. The user is presented with graphical and numerical information about the quality of the solution during the interactive search process to aid him / her make decisions about the search direction.

The integration of GSR with 3T has significant potential for improved system performance. Test results suggest that reductions up to 50% can be achieved in terms of total completion-time and total tardiness metrics. These tests compared the results obtained with a planner using a rudimentary scheduler with that of the integrated 3T-GSR.

Related work Detailed reviews of control of manufacturing systems can be found in (Buzacott & Yao 1986; Gershwin *et al.* 1986). Previous research where explicit consideration is given to recovery from disruptions in schedules includes (Yamamoto & Nof 1985), where a new schedule is generated each time a disruption occurs, and (Bean *et al.* 1991), where recovery from disruption is made during a transient period of time, after which the new schedule matches up with the disrupted schedule. Wu et al (Wu, Storer, & Chang 1991) explicitly consider costs associated with rescheduling jobs before and after their original start times. However, none of this work integrates scheduling with a powerful planner to allow for long-range considerations to be taken into account by the scheduler.

Integrating planning and scheduling has received surprisingly little attention. There are two large efforts currently underway. The first is the OZONE/DITOPS project at Carnegie Mellon University (Smith, Lassila, & Becker 1996). The focus of this work, supported by the ARPA/Rome Laboratories Planning Initiative, is military crisis-action deployment scheduling. However, in this project planning does not mean full-fledged, state-based planning as we do with AP. Instead, planning refers to the preliminary phase of scheduling. This lends itself to problems in which the search-space is well-known and optimization is the important criteria. A second project is an integrated process planning and production scheduling system being developed by Raytheon and Carnegie Mellon (Sadeh *et al.* 1995). Again, the focus is on scheduling rather than planning. The system we propose has more powerful planning capabilities combined with a state-of-the-art scheduling engine.

Crop Planning and Scheduling (CPS)

This module is based on the work in [Leon 1995 and Leon 1996] and deals with the decisions of What, When, Where, and How Much to plant during a given planning horizon. What to plant decisions must select between various plant types (about 15 or more). When to plant consider a planning horizon measurable days. Where to plant is determined by the best growing conditions for each crop and affinity with other plants sharing the same growth chambers. How much to plant is restricted by the maximum planting area in the growth chambers (currently a few hundred square meters) and the size of the planting trays (currently about one square meter). The objective used is a function of the deviations from the ideal reservoir levels. This objective is used as a surrogate measure of the probability of survival. Other considerations are crew menu preferences, nutrition requirements, food stocks, and crew size changes. As a result, the decisions under consideration are non trivial due to the large size of the combinatorial solution space and inherent nonlinearity of the problem.

Interaction layer

Human intervention will be needed during both the planning and the scheduling phases. Human intervention at the planning level is needed to assist in generating and modifying plans and in viewing and comparing plans. Human intervention at the scheduling level is needed to allow the user to set preferences and constraints as well as to allow for changes in scheduled activities. In order to present a consistent interface to the user, all user interaction is with a separate module called the *interaction layer*. This layer maintains a relationship with the user, presenting the appropriate information at the appropriate time and allowing for user intervention into the system. It is not simply a graphical user interface; it is an intelligent agent that may contain a model of the user and their goals and intentions.

An Example Scenario

This scenario will illustrate the interaction among 3T's AP planner, Generic Scheduler Rescheduler (GSR) and Crop Planner and Scheduler (CPS). Figure 2 illustrates the simplified version of a CELSS considered in this example. In this hypothetical scenario the system consists of a crew of 4 people, two autonomous robots, several plant types (e.g., wheat, lettuce, soybean, potato), a food processing machine, limited planting area, limited processing area, and limited holding area.

At the decision time the current state-of-affairs has the following potential goals:

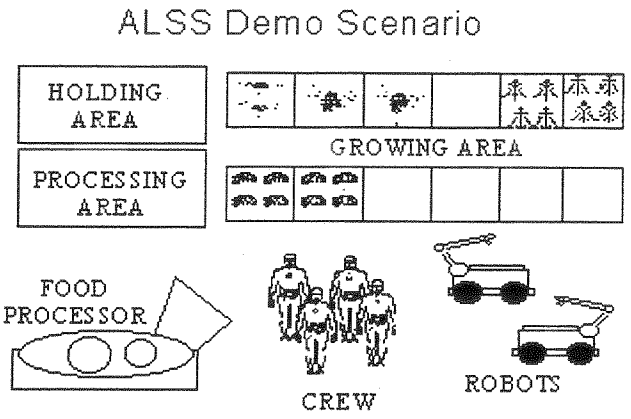


Figure 2: Simplified CELSS system.

- Some planting planting space is available
- A crop of soybean may be ready for harvesting
- A crop of potato is ready for harvesting
- The robots must be serviced within the next couple of days
- The food-processing machine must also be serviced within the next couple of days

Given this situation, the crew desires to generate a detailed schedule of activities that ensures long-term stability. Manual planning and scheduling may be complicated even for this simplistic scenario because: (1) the achievement of each goal requires the execution of numerous tasks, (2) these tasks are inter-related by precedence relations among them, and the usage of scarce resources (i.e., crew, robots, equipment, and space), and (3) the problem is further complicated by the fact that planting decisions will have an long-term effect on the system stability. The proposed architecture is aimed at aiding the decision maker in this kind of scenario.

Given the above state of the world, the crew decides to use the integrated architecture to generate a detailed schedule of crop-related activities and maintenance activities that will ensure the safe and stable operation of the ALSS. Recall that the crew is seeking for recommendations about what to do with the available planting space, soybean and potato harvesting, and robots and food-processing maintenance. In order to generate the detailed schedule, AP will interact with its scheduling (GSR) and crop planning (CPS) modules, in that order, as illustrated on Figure 3.

AP and CPS interaction. In order to decide WHAT, HOW MUCH, and WHERE to plant, such that long-

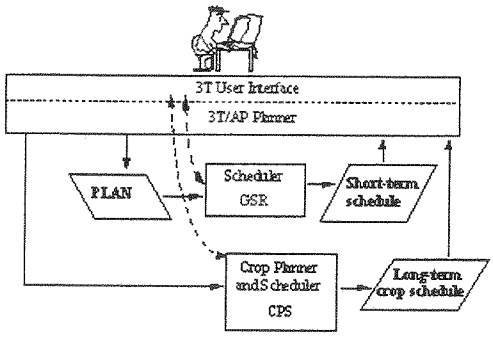


Figure 3: 3T - CPS - GSR Interaction.

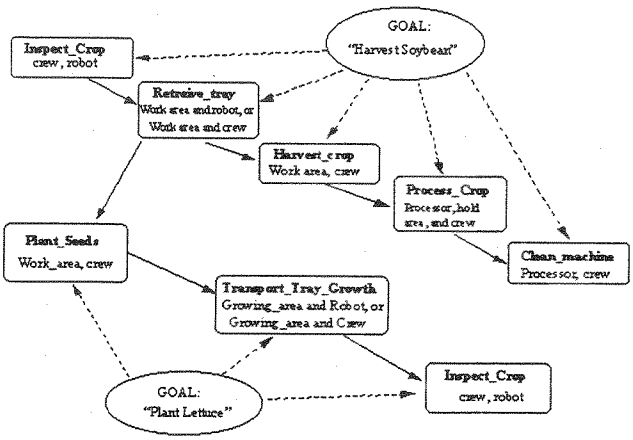


Figure 4: Partial plan generated by AP.

term stability is warranted and the probability of survival is maximized, AP relies on CPS. CPS takes the current state of the system reservoirs (food stores, gases, water), the expected crew profile for the next few months/years, and the state of the in-progress crops, among other system information. For this example, let us assume that CPS recommends, through an interactive session with the decision maker, to plant Wheat, Lettuce and Tomatoes on the available areas. This information (What and how much to plant) is passed to the AP planner. At this point the AP planner has all the goals required to generate a plan of activities.

AP planning

Given the goals "Plant Wheat," "Plant Lettuce," "Plant Tomato," "Harvest Soybean," "Harvest Potato," "Service Robots," and "Service food-processor," the AP planner searches its space-state until finds the tasks that will be passed to the lower levels of the architecture. Figure 4 illustrates the tasks required to accomplish the goals Plant Lettuce and Har-

vest Soybean.

The planner also designates which resources are suitable for the execution of each task and estimated resource consumption. The planner also may require additional time and precedence constraints as specified by operational conditions or the user. Notice that, the planner specifies that the "Retrieve-Tray(Soybean)" task must be accomplished before "Plant-Seed(Lettuce)."

Clearly, there will be significant contention among the activities (or primitives) to utilize the available scarce resources; that is, crew, robots, processing machine, etc. In order to ensure the efficient utilization of the resources AP calls GSR for a detailed schedule and resource-to-task assignment. Figure 5 illustrates the complete plan generated by the AP planner as it is passed to the GSR scheduling module.

GSR Scheduling

In an interactive session between GSR and the decision maker, a detailed schedule containing the exact execution time of each of the tasks is generated. In this schedule, the resources required to execute each task is completely specified. The resulting schedule is one that compromises the total completion time, tardiness of the activities and the decision maker's desires.

From the scheduling perspective, GSR possesses five important characteristics. First, GSR can handle complex precedence relations between activities. Second, GSR resolves alternative resource-group assignments which were only partially specified by the planner. Third, GSR handles multiple-objective explicitly. Fourth, GSR can easily implement resource-time constraints. Finally, GSR can function in a user-interactive mode, or in a fully automated mode.

The current implementation of GSR has a graphical user-interface which allows the user modify the search direction by interactively adjusting the relative weights between performance criteria, and algorithmic parameters. Graphical and tabular displays of the solutions found so far guide the user through the decision process and eventually, in the selection of the schedule to implement. Two output displays are used. Specifically, a Summary Report and a Gantt Chart. The Summary Report form contains a graphical display that allows the user to determine the goodness of a solution in all performance criteria with respect to an "ideal" solution. This report also contains a table with the numerical values for the various performance criteria. The Gantt Chart form displays the activities to be carried out by each resource as a function of time. The user can easily navigate from one form to the next in a windows-like environment.

What-if Analysis and Rescheduling with GSR

The computational efficiency and structure of GSR allows for effective "what-if" analysis and rescheduling. The user could easily modify his / her preference for resource-groups to execute a given activity, or could impose new precedence relationships between tasks.

Execution, monitoring, control and replanning

Once a detailed schedule is generated and approved by the user, AP starts requesting its execution to the lower layers of 3T. AP monitors the execution of the plan. If the system seems to be deviating significantly from the nominal path or new information about events become available, then AP may automatically request for a replanning. The crew can request for replanning at desire.

Current applications to ALSS

We are applying the 3T control architecture (without the scheduling extension) during a 90 day manned test in September 1997 of advanced life support systems for the Lunar/Mars Life Support Technical Program (LMLSTP) at NASA Johnson Space Center (JSC). 3T will be used to control the transfer of product gases (oxygen and carbon dioxide) between multiple gas reservoirs, including a plant growth chamber, storage tanks, the crew habitation module, and an airlock from which the solid waste incinerator draws air and vents effluent. For this application, the planning tier is essential to manage complex system reconfiguration and changes in control strategy required to maintain these multiple gas reservoirs at required levels during a variety of activities including seed germination, plant growth, harvest, and incineration. Even in this constrained application, we have identified a need for the GSR to provide a finer time granularity in the plan (a detailed schedule) and more exact control of start and stop times for activities.

The 3T control architecture also has been selected for controlling computer-controlled machines (robotic and regenerative life support) in the BIOPlex facility to be completed at NASA JSC in 2000. The BIOPlex facility will be a ground-based, manned test facility for advanced life support technology destined for use in lunar and planetary bases, and planetary travel (such as Mars Transhab Project). It consists of five connected modules - two plant growth chambers, a crew habitation module, a life support module, and laboratory. Regenerative life support systems include water recovery, air revitalization, solid waste management, and thermal/atmospheric control. Plant support systems

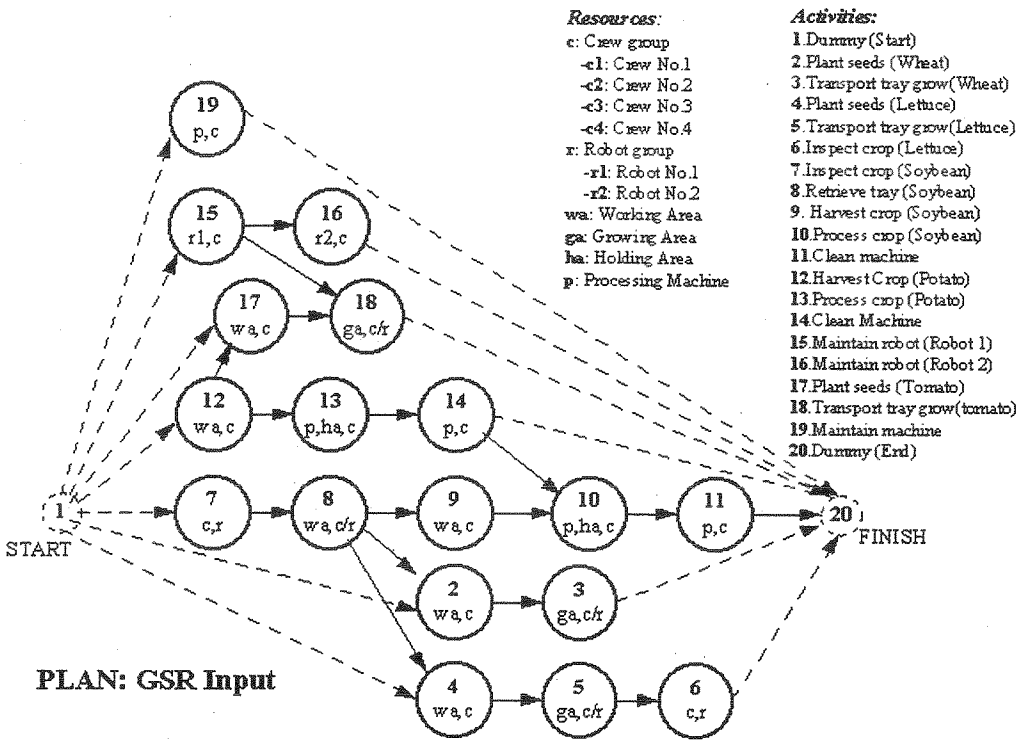


Figure 5: GSR input: the complete plan.

include nutrient delivery, gas management, and thermal/humidity control. Robotic systems include transport, manipulation, and sensor/video scanning. Controlling these heterogeneous systems to maintain food supplies and water and gas reservoirs, while minimizing solid waste reservoirs (inedible biomass and fecal matter) poses a challenging set of problems for planning and scheduling. The planner must balance conflicting system needs and account for cross system coupling, at time scales varying from hours to months. In this facility, human and robots will jointly execute tasks and must coordinate their efforts. A common/shared schedule for both crew and computer-controlled machines is needed to guarantee such coordination. This schedule must be sufficiently flexible to adapt to crew preferences while stable and robust for computer control. An integrated planning, scheduling, and control architecture that includes both fine time grain scheduling and optimization (GSR) as well as long term crop planning (CPS) will be required for BIOPlex.

Concluding Remarks and Future Work

This paper discusses the main characteristics of a CELSS and the requirements for intelligent planning, scheduling, and control architectures. An architecture developed jointly by Metrica Inc. and Texas A&M University for NASA-JSC is described and its functionality illustrated via a scenario. Preliminary testing of different components of the architecture are showing promising results. Future work includes the integration of the crop planning and scheduling module, the development of a coherent and intelligent interface layer, and enhancement of all modules.

References

Auslander, D. M. 1981. CELSS system control overview. In *Controlled Ecological Life Support Systems – Proceedings of First Investigators Meeting (NASA Publication 2247).*

Bean, J.; Birge, J.; Mittenthal, J.; and Noon, C. 1991. Matchup scheduling with multiple resources, release dates and disruptions. *Operations Research* 39(3).

Bonasso, R. P.; Kortenkamp, D.; Miller, D. P.; and Slack, M. 1995. Experiences with an architecture for intelligent, reactive agents. In *Proceedings 1995 IJCAI Workshop on Agent Theories, Architectures, and Languages.*

Buzacott, J., and Yao, D. 1986. Flexible manufacturing systems: A review of analytical models. *Management Science* 32(7).

Colombano, S. 1981. Control problems in autonomous life support systems. In *Controlled Ecological Life Support Systems – Proceedings of First Investigators Meeting (NASA Publication 2247).*

Elsaesser, C., and MacMillan, R. 1991. Representation and algorithms for multiagent adversarial planning. Technical Report MTR-91W000207, The MITRE Corporation.

Firby, R. J. 1989. *Adaptive Execution in Complex Dynamic Worlds.* Ph.D. Dissertation, Yale University.

Gershwin, S.; Hildebrant, R.; Suri, R.; and Mitter, S. 1986. A control perspective on recent trends in manufacturing systems. *IEEE Control Systems Magazine* 1(1).

Hayes-Roth, B. 1995. An architecture for adaptive intelligent systems. *Artificial Intelligence* 72.

Kortenkamp, D.; Bonasso, R. P.; Ryan, D.; and Schreckenghost, D. 1997. Traded control with autonomous robots as mixed initiative interaction. In *AAAI Spring Symposium on Mixed Initiative Interaction.*

Leon, V. J., and Balakrishnan, R. 1995. Strength and adaptability of problem-space based neighborhoods for resource-constrained scheduling. *OR Spektrum* 17(1).

Leon, J. V. 1995. Intelligent planning and scheduling for controlled ecological life support systems. Technical report, NASA Summer Faculty Fellowship Program.

Leon, J. V. 1996. Integration of CELSS simulation with long-term crop scheduling. Technical report, NASA Summer Faculty Fellowship Program.

MacElroy, R. D. 1981. Current concepts of the CELSS programs. In *Controlled Ecological Life Support Systems – Proceedings of First Investigators Meeting (NASA Publication 2247).*

Musliner, D.; ; Hendler, J.; Argrawala, A.; Durfee, E.; Strosnider, J.; and Paul, C. 1995. The challenges of real-time AI. *IEEE Computer* 28(1).

Musliner, D. J.; Durfee, E.; and Shin, K. 1993. CIRCA: A cooperative, intelligent, real-time control architecture. *IEEE Transactions on Systems, Man and Cybernetics* 23(6).

Musliner, D. J.; Durfee, E.; and Shin, K. 1995. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence* 74(1).

Sadeh, N.; Laliberty, T.; Bryant, R.; and Smith, S. 1995. Development of an integrated process planning/production scheduling shell for agile manufacturing. In *Proceedings of the IJCAI-95 Workshop on Intelligent Manufacturing.*

Simmons, R. 1990. An architecture for coordinating planning, sensing and action. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control,* 292–297.

Smith, S. F.; Lassila, O.; and Becker, M. 1996. Configurable, mixed-initiative systems for planning and scheduling. In Tate, A., ed., *Advanced Planning Technology.* Menlo Park CA: AAAI Press.

Wilkins, D. E.; Myers, K. L.; Lowrance, J. D.; and Wesley, L. P. 1995. Planning and reacting in uncertain dynamic environments. *Journal of Experimental an Theoretical AI* 7.

Wu, S.; Storer, R.; and Chang, P. 1991. A rescheduling procedure for manufacturing systems under random distributions. In *Proceedings of Joint US/German Conference on New Directions for Operations Research in Manufacturing.*

Yamamoto, M., and Nof, S. 1985. Scheduling/rescheduling in the manufacturing operating system environment. *International Journal of Production Research* 23(4).

Load Forecasting of a Space-based Telecommunications Network: NASA's TDRSS

Allen J. Levine
Networks Division
NASA GSFC, Code 532.1
Greenbelt, MD 20771
E-mail: Allen.J.Levine@gsfc.nasa.gov

David H. Joesting
AlliedSignal Aerospace
1 Bendix Road
Columbia, MD 21045
E-mail: David.H.Joesting@atsc.allied.com

Abstract

Planning to assure adequate resources exist for low-earth orbit satellite telecommunications through NASA's space-based Tracking and Data Relay Satellite System (TDRSS) is an on-going activity at Goddard Space Flight Center (GSFC). The software tools used in this process were originally developed to analyze, and phase down, an older NASA tracking ground network. The tools have been enhanced over many years with retention of the original ground-based network capabilities. The analysis techniques involve modeling possible network configurations and simulating expected satellite tracking, telemetry, and command (TT&C) telecommunications requirements on a priority basis. The key to generating representative deterministic schedules is simulation procedures that efficiently support complex generic requirements while at the same time generate results that are consistent with what could be expected by the actual network under similar conditions. Over the years this approach has proven itself although there is a bias to overestimate expected loads due to unforeseeable satellite launch slips. This type of architecture can be easily and quickly applied to hybrid space-based and ground-based networks of any practical complexity.

At the Goddard Space Flight Center (GSFC), network capacity planning analyses and capability assessments of low-earth orbit (LEO) satellite telecommunications support have been performed for at least 25 years. These activities originated in support of the Spaceflight Tracking and Data Network (STDN), a NASA network of approximately 20 ground stations. This network was composed of antennas of diverse sizes and encompassed a wide range of radio frequency (RF) bands ranging from VHF to S-band. Since then NASA has expanded to the X- and K-bands with greatly increased data transfer speeds. In the late 1970's in an effort to consolidate operations and reduce costs, while taking advantage of emerging technologies, NASA turned to the Tracking and Data Relay Satellite System (TDRSS). Using an existing application designed for ground station planning through simulation and analysis of spacecraft support schedules, workarounds were developed for these new "ground stations" whose elevations were at geosynchronous altitude. For the most part this worked; however, we found that there were new attributes at play and inherent inefficiencies using the current processes; so new capabilities needed to be added.

Rather than start over and produce new tools from scratch, the Network Planning and Analysis System (NPAS), as the existing system was called, was re-engineered. This integrated set of C and FORTRAN programs initially operated on IBM mainframes and has since been migrated to Hewlett-Packard Series 9000 workstations using X-windows' interfaces. It was originally developed and maintained for NASA by Bendix Corporation, now part of AlliedSignal Aerospace. In the latest version, the NPAS software has been modularized to ease future upgrades. It has retained all previous ground station capabilities, as well as incorporating new relay satellite service operational capabilities and constraints including advanced scheduling algorithms. The compelling reason for keeping the ground network (GN) capabilities in NPAS during the re-engineering was the need to analyze the transition/phase-down from a GN to a space network (SN) which was occurring at that time. Ground station reductions and closures, coupled with the service transition to the relay satellites, needed to be planned carefully in order to minimize impacts to the network's customers. Even after the transition was complete, GN capabilities were retained, because of anticipation of combined GN/SN supports. In recent years, the concept of "service provider" flexibility has become increasingly valuable with the evolution of cross network service provision concepts. It is possible to have a constellation of SN relay satellites in mid-altitudes, and also use cross-links to ground stations or high altitude SN relay satellites for completion of telecommunications links.

The approach that we at GSFC have taken to analyzing a network's forecasted load is a deterministic rather than a statistical or probabilistic one. Using the Goddard Trajectory Determination System (GTDS) for planned orbit computations, a database containing a given network configuration and mission support requirements and constraints, and scheduling algorithms that closely mimic real-world operations, the tools provide valid schedules of communication services for periods of future time. We

have found that the true strength of the deterministic method is its ability to quickly model adverse or exotic requirements, for example, simulating the real-time support of earth land boundary viewing. This is a highly appropriate procedure for investigating the sensitivity of mission requirements satisfaction as network resources are varied. Scenarios defined at any level of support or constraint complexity may be quickly modeled, analyzed and reported with confidence.

With regard to network resources, the physical characteristics of simulation modeling of a single Tracking Data Relay Satellite (TDRS) is somewhat different than the modeling of a ground station. Ground stations typically contain one or more single frequency antennas with both a forward (uplink/command) and return (downlink/receive) capability. Only one user satellite is supported at a time on each antenna. The current version of the space-based TDRS consists of two single access (SA) antennas that each support two different RF telecommunication bands, S-band and Ku-band, both forward and return services. These SAs operate independently of each other and, except in rare situations, a single SA is only used by one user satellite at a time. In addition, a TDRS contains a multiple-access return antenna (MAR) which supports up to 20 return services at a time. (Current TDRSS ground terminal implementation limits each TDRS to 5 MAR services.) The maximum datarate on any MAR is nominally 50 kilobits per second (100 kilobits if the I & Q channels are used at the same time) which is considerably less than on the SA. Lastly, each TDRS has a multiple-access forward antenna (MAF) that is separate from the MAR. The MAF can be used independently of the MAR, however only 1 such service per TDRS is available.

One factor that is somewhat unique to the TDRSS arena is the prototype event with its possible complex assignment of support services. A prototype event is a set of planned telecommunications services from the 3 different TDRS antenna types defined above: SA, MAR and MAF. The services are known as KSAF, KSAR, SSAF, SSAR, SMAF and SMAR. (The first letter refers to the RF band, the second and third defines Single Access or Multiple Access, and the fourth means forward or return.) A prototype event can include any of the three antennas, using any of the services as well as 1- or 2-way tracking services, concurrently and consecutively. Satellites, such as Landsat-5 and the Hubble Space Telescope have reasonably complex prototype events. An example showing the time relationships between services within one possible prototype event is given in Figure 1.

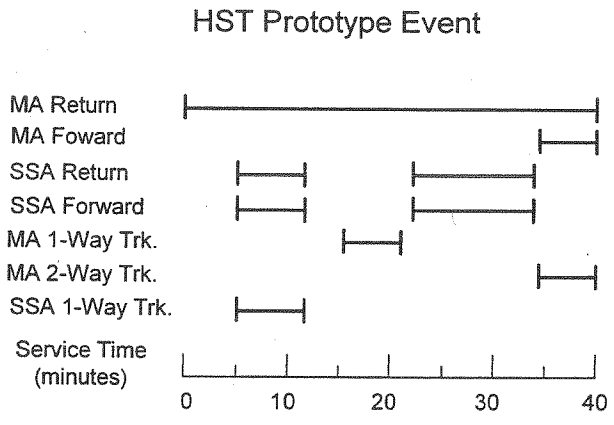


Figure 1

Typically, a TDRS view period of a LEO satellite is relatively long, in most cases approximately 50 to 55 minutes. As a consequence, when one satellite's support requirements conflict with another, the possibility exists of shifting one or more of the services in time within their visibility to resolve any conflicts. Therefore, allowing user requirement definitions of temporal tolerances or windows (within practical limits, of course) is important. Overlapping visibilities by multiple TDRSs provide even more flexibility in resource allocation for any scheduling simulation or even in real operations, if the system takes advantage of it.

Also fairly unique in the SN world are the powered flight launches/expendable launch vehicles, aircraft (powered and balloons), and land or fixed systems that look for long uninterrupted service, even at high data rates (at megabits per second). Support for these types of projects may be beyond the scope of ground stations, not only on a coverage basis, but also because the user ties up a single resource for longer periods of time. This type of support seems to make optimization of a representative schedule easier in the fact that there are fewer combinations of resources. At the same time it can have the opposite effect, this single extended use of resources may make the overall load on the network such that a true optimal solution does not exist (not all users can get all the support they desire.)

In general, when performing capability assessments of ground-based networks, RF constraints take a back seat to geometric line-of-sight restrictions. While sufficient signal strength is important, independent analysis could be performed to determine the feasibility of support. Assuming reasonably capable telecommunications equipment on board the spacecraft and on the ground,

relatively low- or mid-gain antennas with wide beamwidths can be placed on the spacecraft, minimizing RF blockage and attitude constraint problems. This is not, however, true for services through a geosynchronous relay satellite. Although omni-directional antennas can be used for some telecommunications support, especially at low data rates using the TDRS Single Access (SA) antenna, in many cases a high-gain (directional or pointing) antenna is required by the user spacecraft. Thus antenna blockages and attitude constraints are very important factors in determining the ability of a space-based network to provide service and their definition becomes a significant requirement in performing a meaningful capability assessment.

Telecommunications support requirements can be defined specifically or generically. In NPAS, both specific and generic requests are processed. A specific request is one that defines specific services, times and resources (for example, 20 minutes of SSA forward and return service on a particular TDRS at 1200 GMT). A generic request allows the system some latitude to do the selecting (for example, between 15 and 20 minutes of both SSA forward and return service by any TDRS once per day). A further extension of a generic request is its ability to define recurring requirements, that is, one request statement defining multiple requests (for example, once per orbit, or twice per day separated by at least 12 hours). A robust model would be able to include all of the above defined requirements requests.

One unique situation that manifested itself in the early days of SN analysis that hadn't appeared in the earlier GN studies involves the "per orbit" requirement. An orbit is generally defined as one complete circuit around the earth; early convention typically defined the start and end points as being the successive ascending equatorial crossing. A "per orbit" service requirement definition for a network of ground stations is easy, but with geosynchronous relay satellites, it can become a problem, if long visibilities are required. These relay satellites, like ground stations, move with the earth's rotation; user spacecraft do not. Taking a simple case of a user satellite with a 95 minute orbital period, with no precession, would see the earth shift in one orbit almost 24 degrees. The TDRS visibility therefore intersects and crosses the orbit boundaries. Many times a customer will request "2 TDRS events per orbit", thinking one event per TDRS, that is, one TDRS stationed over the West Pacific (TDW) and one over Brazil (TDE). What they are really saying is "I want two TDRS events every TDRSS viewing cycle." A TDRSS viewing cycle is, however, longer than an orbit since over the course of an orbit, the TDRS has moved approximately 24 degrees. This equates to 6 minutes more; the cycle is 101 minutes. This has important implications for on-board data storage management, and also the estimate on the number of events per day requested (e.g., 14 versus 15 for "I per orbit", 28 versus 30 for "2 per orbit"). The current challenge is to determine a good convention to cover the possible and probable relay satellite constellations. The one typically used in NPAS is the longitudinal midpoint between TDE and TDW in the Zone of Exclusion, which is the region of no coverage by TDE and TDW of satellites under 1200 Kilometers altitude and with low to mid orbital inclination. (For a network of relay satellites at lower altitudes, the orbit cycle convention will probably be better suited.)

A number of solution methods to the telecommunications service scheduling problem exist, most of which involve a priority or service request processing order. The two methods are not necessarily the same. (The priority order indicates a hierarchy while a processing order can allow for adjustment.) Furthermore, one can take the defined requirements and expand them up front (for all users), then "fit" the resultant services all together, like a puzzle, or expand each requirement one at a time and place it in the "schedule" as it is processed. NPAS uses the latter because this provides the maximum flexibility by retaining generic definitions including any level of complexity "up to the last second". The former method appears to have the advantage of simplifying schedule adjustments since all requests are defined prior to "scheduling," however, for all practical applications this method is computationally expensive. In any realistically representative NASA Space Network support scenario, the number of possible puzzle pieces, (time dependent pieces in this puzzle change shape each time related piece/service is "fitted") is enormous. This becomes a substantially larger problem if multiple service prototype events are supported.

Within the NPAS standard priority scheme, prototype event scheduling is performed based on longest service length first and then station priority (if any). In order to simulate the complex scheduling optimization that may occur within any single user's own scheduling operations, a special "geometric optimization algorithm" may be invoked for a set of generic requirements. This process is designed to maximize the total scheduled time for that generic request. The process starts with an initial greedy schedule with local optimal solutions. If the total support is less than the desired minimum, then a limited depth first search with backtracking process is invoked. During its operation, it considers local sub-optimal solutions in order to generate a better global solution. The limit to which the depth of searching is allowed may be controlled by the modeling analyst based on time considerations.

Periodically, an indirect route is taken in examining the ability of the SN to provide service to a prospective new customer. Analyzing the resource "free" time, or unutilized time, in terms of service gap time durations and frequency, often provides valuable insight to the ability of the network to service that additional customer. If a new project comes along and indicates a desire for services that do not impact existing customers, then a quick analysis can be performed to determine whether or not the new customer's needs can be met. Even with the possibility of impact, an experienced analyst can make preliminary judgements as to the probable success of the additional services.

An important consideration is the resultant accuracy of any simulated operational schedule. It would be possible to develop computer schedule solution procedures using generic satellite support requirements that would in effect be too accurate. The purpose of the network loading analyses, whether SN or GN, is to simulate the performance of the real network with expected resource configurations and mission support requests. For the most part in the SN, the real operational environment consists of individual satellite projects expanding their own requirements into events with fairly specific start times and durations for each services. These sets of events are sent in on a weekly basis for a support week up to 3 weeks in the future. Conflicts between any two users are settled manually by operations controllers, typically on a priority basis. Rejected events are returned to the satellite project for modification and resubmittal. Due to the different responses by the projects based on their own internal priorities, this is an iterative process which cannot be directly simulated.

The NPAS process has shown itself to fairly closely mimic the results of that iterative process without introducing too much optimization that would bias the results towards overestimating either the network's capacity or an individual satellite's expected support. However, there are other times when a near optimal schedule may be required. Instead of introducing the up front expansion of the users' requests with that procedure's known computationally intense requirements, a second approach has been taken in the NPAS. The resultant schedule from a normal conflict free NPAS simulation, which represents a good operational schedule typically well within 95% of the true optimal schedule, is taken as the starting point for a optimization run. All missions' requirements are expanded and missions requirements that are not 100% satisfied in the starting schedule are revisited. Event time movements of all missions on required network resources are attempted within tolerances. Tolerances here are determined by intersections of the original visibilities and the mission's stated generic requirements. Backtracking to predefined levels of complexity is used in this process. Based on the amount of computer time available, a better to near optimal representative schedule is obtained.

In the capacity planning process, it is important to analyze not only individual network configurations and mission support profiles, but to also obtain an understanding of the sensitivity of the resultant loading to the various support parameters. One area of concern is that the results of solution schedules are somewhat dependent on the relative alignments of the user satellites in their orbits. NPAS capability assessments typically begin with a coverage variability analysis portion that attempts to identify worst, best, and average (expected) cases. This is accomplished by the use of Monte Carlo simulation methods, which vary the defined mission orbital parameters in a hundred or more simulations. The satellite orbit profiles from each of the three cases are made available for use based on the purpose of the assessment.

In ground station analyses, downtimes for system maintenance can be modeled as well as failure modes. In general, however, failures can be ignored or de-emphasized since repairs can be made. With a relay satellite, it is more difficult. When a hardware failure occurs onboard, there is no way currently to effect a repair, redundancy becomes critical. As a result, failure modeling and replenishment needs analysis are needed to complete the work. The TDRS reliability models are generally analyzed at the subsystem level, although the actual reliabilities are defined down to the component level. The four subsystems defined are the Payload, Bus, SAC East, and SAC West. The first 2 subsystems are critical items for any telecommunications services to be provided, so they are single points of failure, although there are redundant components and noncritical elements within each. The 2 SACs are the 2 Single Access antenna systems, and therefore are single points of failure only for the services provided by only that antenna. Therefore, it is possible for a TDRS to have only one SA antenna in the modeling. Currently, the analysis does not go down to the service level, although in real life the TDRS SA systems are used even when some services are not available. TDRS-3, for example, has only one fully functional SA antenna system, the other SA has S-band services and Ku-band return services, the Ku forward being nonfunctional. This capability is currently being implemented in the modeling; the only question remaining is how to present the results and how to judge the viability of TDRS service. Part of the decision is a management one - is the service cost effective?

Historically, load forecasting of the SN has been reasonably accurate for near term planning; long term forecasting was somewhat uncertain. However, with the changing environment, with NASA becoming more focused in its overall planning and improvements in reliability in its systems, forecasting is likewise improving. (Refer to Figure 2.) In the earlier days, with a more robust and optimistic budget, the representative NASA enterprises projected a number of study missions, each of which was felt had a reasonable chance of acceptance. At that time, it was more difficult to predict the number and types of missions to receive commitments due also to the optimism and also due to relative inexperience in forecasting. For the Space Network, a drastic difference between prognostication and reality occurred when the Challenger accident happened. Launches by the Space Shuttle stopped for almost 3 years, commercial satellites (with possible SN customers) were taken off the Shuttle manifest, and a TDRS was lost, being a passenger on that ill fated launch. As it turned out, most of the projection was just delayed approximately 3 years, so a slide could actually be applied to the projection curve.

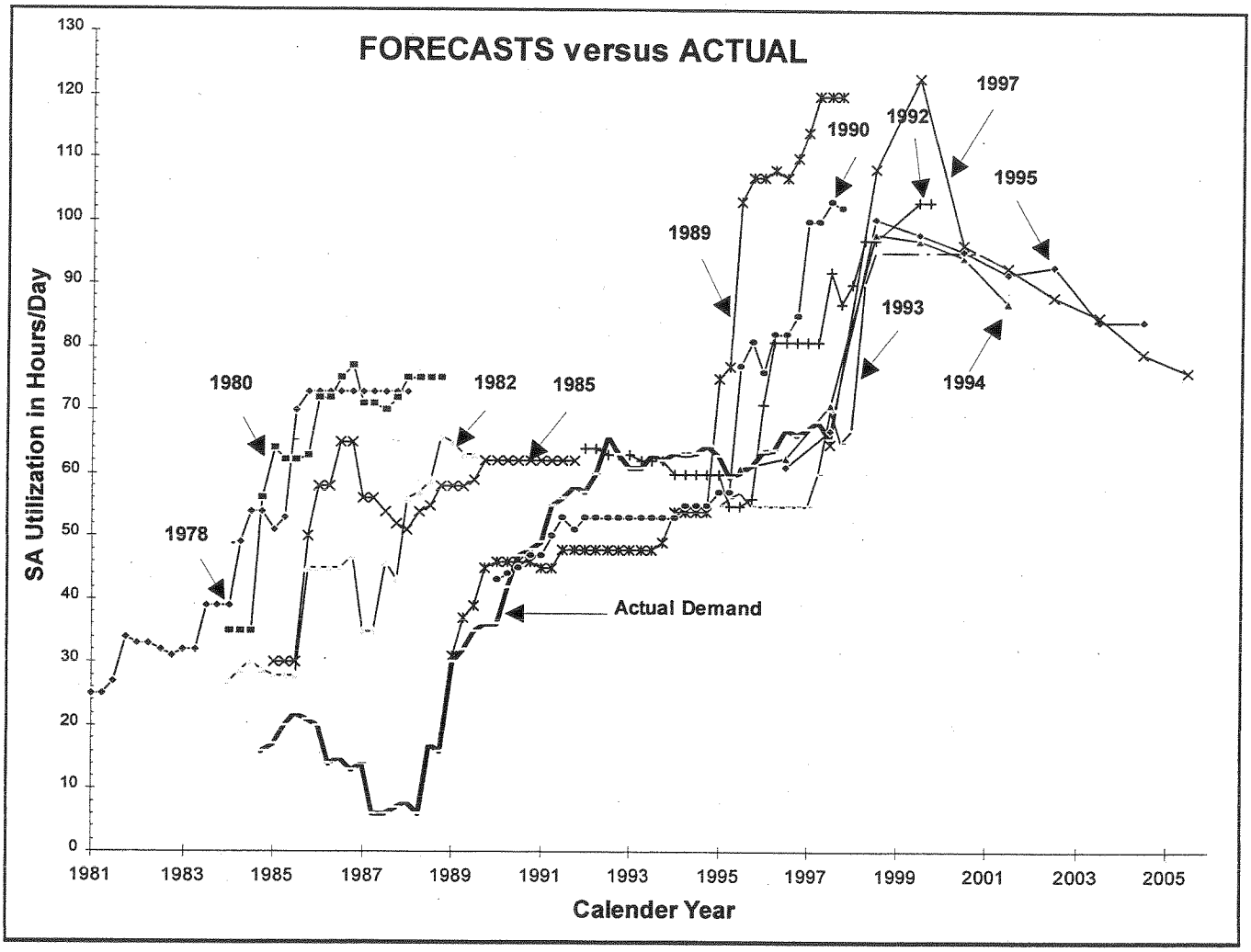


Figure 2

One of the interesting trends in satellite communications capacity planning covers the ground and space networks teaming scenarios. The newer customers are starting to look at direct downlink science/telemetry data, basically getting the principal investigators' data directly, or almost directly, to them through ground links, while using the

almost continuous visibility capabilities of space-based stations for spacecraft monitoring and control. Lower data rates (for spacecraft telemetry) and periodic (possibly on-demand) commanding and/or tracking through the TDRSS have been shown to be viable due to the currently forecasted light loading on the MA system.

In addition to this type of analysis, there is the further step of performing GN verses SN trade-off studies. In today's competitive environment, this type of analysis is critical. Obviously, a consistent set of metrics is required; a tool such as NPAS is needed to perform the appropriate analyses. The only other requirement is obtaining from the prospective customers a consistent sets of communications requirements for each support network. It would seem to be a simple matter, however, some problem has been experienced with one or two customers (who may have had a predisposition toward one type of network). Sometimes, more "digging" is needed to provide an objective answer.

Given the changing environment, proven and adaptable tools are required to provide proper capacity planning analyses and capability assessments for telecommunications support. Recognition of the difference between space-based and ground-based tracking systems is necessary as well as the ability to team their support when appropriate. Melding the capabilities into a cohesive model with appropriate analysis is the key to performing effective assessments and recommendations for successful support of all customers.

References

Stern, D. C., Levine, A. J., and Pitt, K. J. 1994. Accuracy Analysis of TDRSS Demand Forecasts. In proceedings of the Third Symposium on Space Mission Operations and Ground Data Systems, 331—318, Greenbelt, MD: NASA Conference Publication 3281.

Simons, M., and Larsson, G. 1994. Analysis of Space Network Loading. In proceedings of the Third Symposium on Space Mission Operations and Ground Data Systems, 1071—1077, Greenbelt, MD: NASA Conference Publication 3281.

Brase, J., and Burns, M. 1997. TDRS Space Network Reliability and Availability Modeling Status Report. Stanford Telecom.

Large-Scale Planning Under Uncertainty: A Survey

Michael L. Littman and Stephen M. Majercik
Dept. of Computer Science
Duke University
Durham, NC 27708-0129
{mlittman,majercik}@cs.duke.edu

Abstract

Our research area is planning under uncertainty, that is, making sequences of decisions in the face of imperfect information. We are particularly concerned with developing planning algorithms that perform well in large, real-world domains. This paper is a brief introduction to this area of research, which draws upon results from operations research (Markov decision processes), machine learning (reinforcement learning), and artificial intelligence (planning). Although techniques for planning under uncertainty are extremely promising for tackling real-world problems, there is a real need at this stage to look at large-scale applications to provide direction to future development and analysis.

INTRODUCTION

Planning—making a sequence of choices to achieve a goal—has been a mainstay of artificial intelligence (AI) research for many years. Traditionally, the decision-making models that have been studied admit no uncertainty: every aspect of the world that is relevant to the generation and execution of a plan is known in advance. In contrast, work in operations research (OR) has focussed on the uncertainty of the effects of actions, but uses an impoverished representation that does not capture relationships among states for specifying and solving planning problems.

The area of planning under uncertainty in large domains explores a middle ground between these two well-studied extremes with the hope of developing systems that can reason efficiently about plans in complex, uncertain applications. This paper reviews some of the most recently developed techniques that may scale well to large, real-world domains.

We feel that the potential benefit of applying such methods to planning problems in the space program is significant. We suggest two NASA activities in particular that would benefit greatly from the development of successful techniques for large-scale planning under uncertainty: scheduling the Deep Space Network (DSN), and on-board planning for autonomous spacecraft.

The purpose of the DSN is to support both unpiloted interplanetary spacecraft missions and radio and radar astronomy observations in the exploration of space (Chien, Lam, & Vu 1997). Planning to fulfill DSN service requests from tens of projects with varying priorities requires the consideration of thousands of possible tracks using tens of antenna resources and hundreds of subsystem configurations. The basic scheduling task is enormously complicated and made more difficult by three characteristics of DSN scheduling described by Chien, Lam, & Vu (1997):

1. The priority of a tracking request is dynamic and can change depending on the amount of tracking a project has received so far in a given time period.
2. Subsystems needed for the execution of tracks are shared by each Signal Processing Center.
3. Projects may request more tracking time than absolutely necessary and specify the absolute minimum; in these cases, the planner has the option of reducing the tracking time allotted to the request if doing so will remove a resource conflict.

There are also significant elements of uncertainty in the scheduling task (Chien *et al.* 1996; 1997):

- The needs of projects—and, thus, plan goals—change over time. Before (or even during) a track, a project may submit a request to add various services to the track.
- Equipment availability may change due to failure, maintenance or recalibration requirements, or preemption by a track with higher priority.
- Changing weather may require changes in an existing tracking schedule.

The system must respond to changing circumstances quickly and efficiently with minimal disruption.

This problem contains many elements targeted by research in planning under uncertainty in large domains. First, the sheer size and complexity means that algorithms designed to work on "toy" problems may not be adequate for this real-world

problem. Second, the elements of uncertainty cited above mean that a successful plan will not be a fixed list of events, but will include contingencies for dealing with various eventualities; for example, the schedule could include alternate assignments that should be made if a new high-priority job were to appear at various points in the future—this would make it unnecessary to begin to reschedule from scratch in this circumstance (Drummond, Bresina, & Swanson 1994). For these reasons and others, a successful planner must take uncertainty into account.

The importance of on-board planning for autonomous spacecraft is emphasized by NASA's New Millennium program, the objective of which is to develop and validate new technologies that will both reduce mission costs and increase mission quality (Muscettola *et al.* 1997). The first New Millennium mission—Deep Space 1—features a Remote Agent autonomy architecture, one of whose components is a Planner/Scheduler, which translates a set of high-level mission goals specified by scientists and engineers into a sequence of low-level commands (Muscettola *et al.* 1997).

The necessity of autonomous spacecraft for deep space missions is made apparent by the extreme distance of the missions' targets, the impossibility of hands-on troubleshooting or maintenance, and the difficulties imposed by light-time delayed communication (Chien *et al.* 1997). The task of the planning component for the required autonomy technology is complicated by limited sensing capabilities, uncertain effects of actions (due to system failures or unanticipated environmental factors), and the fact that the system must construct plans without knowing with certainty the future state of the spacecraft (when the plan will begin executing). Furthermore, an autonomous spacecraft needs to be able to handle unexpected events intelligently and without lengthy deliberation. We will address these issues further in the section on partially observable Markov decision processes.

In the remainder of the paper, we describe recent developments that we feel may have an impact on solving this type of problem. The following section describes work on algorithms for planning problems formalized as Markov decision processes (MDPs), a model developed in the OR community. While MDP-related algorithms are quite mature and have been used in some fielded systems, standard treatments of MDPs use an unstructured representation of states and actions that cannot scale to many large real-world problems. In later sections, we describe some recent work geared to planning using richer state representations, and we describe planning algorithms that use richer representations of both states and actions. We also describe algorithms that address another limitation of MDPs, namely their assumption of "complete observability" during decision making. We conclude with a description of our current research efforts.

MARKOV DECISION PROCESSES

The MDP model is a formal specification for planning under uncertainty originally developed in the OR community in the late 50s and early 60s. The foundational work was done by Bellman (1957) and Howard (1960), and included a formal description of the model and basic results such as the existence of optimal solutions and algorithms for finding them. These algorithms still form the basis of nearly all the current approaches to solving MDPs.

In MDPs, an *agent* has the task of making decisions to solve some planning problem. The agent is embedded in its task environment and must decide on an action to take based on the current state of the environment. The actions it selects result in some immediate cost or benefit to the agent depending on the environment's state and can also change the environment according to its *transition model*. An intelligent agent will select actions that maximize its net benefit over a sequence of interactions with the environment.

This model presents the problem of intelligent behavior in a microcosm; the agent has a concrete measure of the success of its actions and must plan ahead to maximize its success. Compared to the larger problem of creating a broadly intelligent agent, the model includes various simplifications that make it tractable: the state space of the environment is typically finite, as is the set of possible actions the agent can choose from; time is discrete; costs and benefits are additive and not time dependent; and everything that is relevant to the decision-making problem is evident to the agent (i.e., there is complete observability—the Markov property holds for observations).

For MDPs with up to a million states or so, efficient implementations of the classic algorithms—value iteration, policy iteration, linear programming, and modified policy iteration—can be used to find optimal plans (or, policies, more precisely). Many practical problems in OR have been formalized this way and are being solved in commercial settings; Puterman (1994) describes the basic algorithms and cites applications in blast-furnace maintenance, bus-engine replacement, queuing, scheduling, fisheries management, and many others.

The DSN scheduling problem can be modeled as an MDP by creating a state for each combination of current project needs, available resources, and existing assignments. The actions create new assignments of tasks to resources and the stochastic transition model captures the arrival of new requirements and achieved needs. Similarly, planning problems for autonomous spacecraft can be mod-

eled as MDPs in which the states capture all relevant aspects of the environment, the long and short-term goals of the mission, and the status of the spacecraft itself. The actions are the moment-to-moment decisions that the spacecraft can make for itself: firing thrusters, extending antennae, radioing home for help. The transition model describes the "physics" of the interaction between spacecraft and environment, using randomness to capture aspects of the environment that cannot be reliably predicted given the state representation of the system.

Even a modest-size version of these MDPs would likely contain trillions and trillions of states, making classical algorithms practically useless. In the following sections, we present a number of approaches that have been suggested for attacking MDP problems of this scale.

One concept that is central to both classical and recent MDP algorithms is that of a *value function*. Value functions map states of the MDP to a measure of how good it is for the agent to be in that state; they have several attributes that make them invaluable in planning under uncertainty. First, like evaluation functions in game-tree search, they can be used to guide the agent's choice of action using a simple one-step look-ahead scheme; the agent considers each possible action and chooses the one that leads to states with the highest expected value according to the value function. In fact, a standard result is that there is always an *optimal* value function that will guide the agent to making the best possible choice. Second, unlike a simple plan (fixed sequence of actions), behaving according to a value function is robust under uncertainty—at each moment, the agent chooses the best action for the state that it is actually facing. And, third, approximately correct value functions can be improved iteratively, making them easy to use in algorithms. Nearly every algorithm for planning under uncertainty uses a value function in some form.

Several AI researchers have addressed the issue of large-scale MDPs by creating variants of the classic algorithms that use heuristics to make more efficient use of computational resources. Prioritized Sweeping (Moore & Atkeson 1993) maintains an estimate of the optimal value function and uses a rule of thumb to predict when updating the value function for a particular state is likely to be important for improving the approximation. Real-time dynamic programming (Barto, Bradtke, & Singh 1995), or RTDP, attempts to find a good approximate policy quickly by focusing value-function updates on states that are likely to be visited; this approach is sometimes also called the *reinforcement-learning* approach (Kaelbling, Littman, & Moore 1996). *Envelope* methods produce a good partial policy by explicitly identifying states that are likely to be visited and solving a smaller MDP (Drummond & Bresina 1990; Tash & Russell 1994; Dean *et al.* 1995).

VALUE-FUNCTION APPROXIMATION

In the standard MDP approaches, states are represented as being completely unrelated objects. In many domains, states can be described in such a way that "similar" states have similar representations. In the autonomous spacecraft example, it is clear that a natural state representation would be one that explictly notes spacecraft location as distinct from the current objective. Therefore, simply on the basis of the "name" of the state, it is apparent that the two states could be treated similarly for planning.

This insight can be exploited by exchanging the classical table-based method for representing value functions in MDP algorithms for one that uses a function approximator (for example, a neural net) to map state-description vectors to values. A wildly successful example of this is TD-Gammon (Tesauro 1995); this work uses gradient descent and temporal-difference learning (Sutton 1988) (roughly a variant of RTDP) to train a neural-network value function to play backgammon at the level of the best human players. Playing a good game of backgammon is an interesting example of planning under uncertainty because of the impact of the dice rolls and the other player's moves on state transitions.

Several other applications have been developed using this same basic approach, including a controller for a bank of elevators (Crites & Barto 1996), a system for making cellular-phone-channel assignments (Singh & Bertsekas 1996), and a job-shop scheduler for space-shuttle payload processing (Zhang & Dietterich 1995). These commercially relevant applications exhibit a great deal of uncertainty, an astronomically huge state space (10^{20} and beyond), and have been studied closely enough that human-engineered policies are available for comparison. In each case, the automatic planning systems based on value-function approximation result in policies superior to the prior state of the art. We see no fundamental obstacles to applying value-function approximation to a broader array of problems including additional space-related applications.

Recent projects have begun to shed light on the practical and theoretical guarantees that can be made when using value-function approximation. Boyan & Moore (1995) show that representing value functions by neural networks need not behave well. Sutton (1996), however, provides a counterpoint that shows that an appropriate choice of training methodology can improve behavior sig-

nificantly. Positive and negative theoretical results have been derived for gradient-descent methods (e.g., neural networks) and averaging methods (e.g., nearest neighbors) (Baird 1995; Gordon 1995; Tsitsiklis & Van Roy 1996b; Tsitsiklis & Van Roy 1996a). At present, this class of algorithms has been the most successful for solving large, practical planning problems and work continues in this area.

PROBABILISTIC PLANNING

Value-function approximation attempts to exploit structure in the state space (and the value function), but treats actions as black-box transformations from states to probability distributions over states. A promising alternative is to use symbolic descriptions of the actions to reason about entire classes of state-to-state transitions all at once. This is the approach taken in classical AI planning (McAllester & Rosenblitt 1991), and it can be a good deal more direct, and therefore more computationally efficient, than RTDP plus value-function approximation.

The classical view of planning ignores uncertainty. In the STRIPS representation (Fikes & Nilsson 1971), for example, an operator has a list of preconditions that must be satisfied before the operator can be applied. But, when these conditions are met and the operator is applied, the effects of the operator—specified as a list—take place with certainty. Uncertainty can be introduced gently into the STRIPS representation by assuming a deterministic domain with a small amount of "external" randomness (Blythe 1994). A number of researchers have explored the problem of planning given more general representations of stochastic operators (Goldman & Boddy 1994; Kushmerick, Hanks, & Weld 1995; Boutilier, Dearden, & Goldszmidt 1995).

The BURIDAN system (Kushmerick, Hanks, & Weld 1995) exploits a general representation for stochastic STRIPS operators and extends partial-order planning to stochastic domains. Its representation can express arbitrary MDPs, sometimes logarithmically more compactly than traditional OR representations. Similar to traditional deterministic planners, the plans found by BURIDAN are simple sequences of actions. C-BURIDAN (Draper, Hanks, & Weld 1994) extends the BURIDAN system so that the plan representation is more powerful; it can express contigent execution of plan actions, although it is still less powerful than a policy-type representation.

Another area of interest is in solving MDPs expressed in a compact STRIPS-like representation using adaptations of classic MDP algorithms. Boutilier, Dearden, & Goldszmidt (1995) show how the policy-iteration and value-iteration algorithms can be adapted to manipulate compact representations; Boutilier & Dearden (1996) extend this to deal with approximations of the value function. Dearden & Boutilier (1997) adopt the view that value-function approximation is a type of "abstraction," the form of which can be derived automatically from a propositional representation of the planning problem.

Although some promising algorithms have been described in the past few years, we are only just beginning to understand the computational properties of this class of problems. Littman (1997) provides some basic complexity results for probabilistic planning and also shows that most natural compact representation schemes are equivalently compact (to within polynomial factors). Goldsmith, Littman, & Mundhenk (1997) directly address the problem of finding compact representations of plans in stochastic domains; although most problems are computationally intractable, there are some ways of formalizing the problem that may be amenable to heuristic approaches. In a later section, we sketch some of our early work in using these insights to design a new type of planner for large-scale stochastic domains.

PARTIALLY OBSERVABLE MDPs

In real applications, especially those that involve physical devices whose effects on the agent's environment are uncertain, it is often impossible for the decision-making agent to base its choices on the true state of the world; in general, there will always be aspects of the world that are not directly or instantaneously accessible to the agent's sensors. Autonomous spacecraft, for example, are situated in unknown environments, possess limited sensing capabilities, and have a repertoire of actions whose effects are uncertain due to possible system malfunctions or unanticipated environmental factors. In this type of situation, value-function-based algorithms for planning do not work properly. Partially observable Markov decision processes (POMDPs) model the situation in which the agent must cope with uncertainty in its estimate of the current state (Lovejoy 1991; Cassandra, Kaelbling, & Littman 1994).

In general, the POMDP framework extends MDPs to a much wider range of potential applications. However, the resulting problems are often considerably more difficult to solve. New exact algorithms have been designed that can solve larger and more complex problems than those described in the OR literature (Littman, Cassandra, & Kaelbling 1996; Cassandra, Littman, & Zhang 1997); however, even these algorithms can run into difficulty finding optimal solutions to problems with more than a dozen or so states.

Algorithms dependent on heuristics (Washington 1996; Hansen 1994), approximations (Littman,

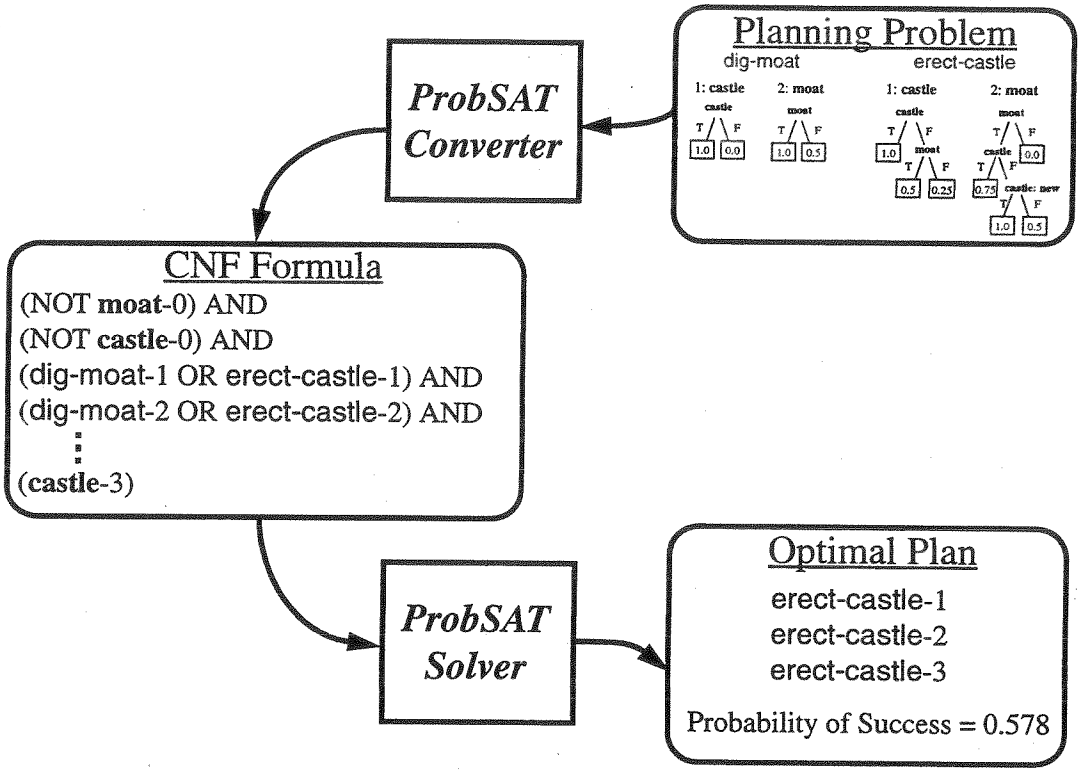


Figure 1: Block Diagram Illustrating the Steps of Our Planner

Cassandra, & Kaelbling 1995; Parr & Russell 1995; Simmons & Koenig 1995), and propositional representations (Draper, Hanks, & Weld 1994; Boutilier & Poole 1996) are being developed and have exhibited improvements in problem size and solution speed over exact approaches. This seems to be an area of intense interest, and many new approaches are being devised, although few of these have been rigorously tested at this time.

CURRENT RESEARCH

Presently, we are exploring an alternate approach to planning under uncertainty that attempts to combine the positive attributes of the various approaches described above. Our system is applicable to probabilistic planning problems, both MDPs and POMDPs, specified in a compact form. It is designed to find plans that maximize the probability of achieving a goal, but under constraints on the size and representation of the plan.

Our approach is inspired by the SATPlan planner of Kautz & Selman (1996). SATPlan uses a propositional logic problem encoding and stochastic search to solve very hard deterministic planning problems as much as an order of magnitude faster than the next best planning system. Briefly, SATPlan converts a deterministic planning problem to a Boolean satisfiability problem by constructing a CNF Boolean formula that has the property that any satisfying assignment to the variables in the formula corresponds to a valid plan—one that results in achieving the goal. This is done by making satisfiability equivalent to the enforcement of the following conditions:

- the initial conditions and goal conditions hold at the appropriate times,
- the existence of an effect at time t implies the disjunction of all actions that can produce that effect at time $t-1$,
- the use of an operator at time t implies the existence of its preconditions at time $t-1$, and
- conflicting actions are mutually exclusive (Kautz & Selman 1996).

The satisfiability of the resulting CNF formula is determined using WalkSAT, a generic satisfiability algorithm based on random hill-climbing.

To adapt this method to probabilistic planning, we make a distinction between *choice variables*, which encode the possible plans, and *chance variables*, which encode the uncertainty in the probabilistic planning problem. Choice variables, like all the variables in a SATPlan encoding, can be arbitrarily set to `True` or `False` in the search for a satisfiable assignment. The truth assignment of chance variables cannot be arbitrarily set; each such

variable is True with a certain probability and, together, the chance variables determine the probability that a given truth assignment for the choice variables (i.e., a given plan) will actually lead to a satisfying assignment (i.e., reach a goal).

Given a CNF formula with chance variables, we wish to determine, not merely the satisfiability of the formula (as in SATPlan), but rather the assignment of choice variables that has the highest probability of producing an overall satisfying assignment. Such an assignment to the choice variables maximizes the probability of reaching a goal. This means that, for each setting of the choice variables (each plan), we must find all possible "assignments" to the chance variables that produce an overall satisfying assignment, and sum the probabilities of these probabilistic assignments to determine the probability of success for that plan. The computational complexity of this problem is NP^{PP}-complete (Goldsmith, Littman, & Mundhenk 1997); thus, the problem of finding the optimal restricted-size plan for a finite-horizon POMDP or MDP in compact representation is likely more difficult than an NP-complete problem like finding the optimal finite-horizon deterministic plan, but likely easier than an EXPSPACE-complete problem like finding the optimal *unrestricted* plan for a finite-horizon POMDP in compact representation (Goldsmith, Lusena, & Mundhenk 1996).

Our technique for solving these probability-extended satisfiability problems is based on the Davis-Putnam procedure for determining satisfiability (Davis, Logemann, & Loveland 1962) and can be envisioned as constructing a binary tree in which each node represents a choice variable or a chance variable, and the two subtrees represent the two possible remaining subproblems given the two possible assignments to the parent variable (if the parent is a choice variable) or the two possible outcomes (if the parent is a chance variable). Clearly, it is critical to construct an efficient tree to avoid evaluating an exponential number of assignments. As in the Davis-Putnam procedure, we do this by selecting for the next variable in the tree, whenever possible, a variable that appears by itself in a clause, or a variable that appears in only one sense (always negated or always not negated) in the formula. We also remove variables which become irrelevant as the tree is constructed. When no such obvious choice is possible, we choose the variable that satisfies the most clauses in the remaining subproblem.

We summarize our current approach to planning under uncertainty in Figure 1, which shows a solution to a problem described by Goldsmith, Littman, & Mundhenk (1997).

The basic insight of our approach—solve a probabilistic planning problem by converting it to an equivalent problem on a Boolean formula—has a number of attractive properties. By casting the problem in a generic format we can take advantage of proven algorithmic techniques from a number of areas of research. Our current solver already incorporates successful ideas from dynamic programming and Boolean satisfiability; future versions will also draw on the areas of AI planning, belief networks, reinforcement learning, and stochastic search.

CONCLUSIONS

Over the past few years, powerful formal models of planning under uncertainty have been explored and scores of new algorithms for planning with these models have been devised. The models appear to be useful for expressing real-world, practical problems and the algorithms show a great deal of promise for solving these problems.

One component that is sorely lacking in this picture is substantive contact with real applications; many algorithms have been tested only on tiny "proof-of-concept" problems and no information is available on how the algorithms scale up. It is critical at this stage to make a serious effort to apply these nascent technologies to real-world problems; only through contact with applications can we hope to focus our research effort in the most promising and productive directions. We believe that planning problems in the space industry would provide an excellent test bed for algorithms for planning under uncertainty that aspire to be effective in real-world domains. The interaction between emerging techniques and real problems will spur the development of more efficient and practical systems as well as contribute to the solution of some important problems.

References

Baird, L. 1995. Residual algorithms: Reinforcement learning with function approximation. In Prieditis, A., and Russell, S., eds., *Proceedings of the Twelfth International Conference on Machine Learning*, 30–37. San Francisco, CA: Morgan Kaufmann.

Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72(1):81–138.

Bellman, R. 1957. *Dynamic Programming.* Princeton, NJ: Princeton University Press.

Blythe, J. 1994. Planning with external events. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, 94–101.

Boutilier, C., and Dearden, R. 1996. Approximating value trees in structured dynamic programming. In Saitta, L., ed., *Proceedings of the*

Thirteenth International Conference on Machine Learning.

Boutilier, C., and Poole, D. 1996. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1168–1175. AAAI Press/The MIT Press.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 1995. Exploiting structure in policy construction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1104–1113.

Boyan, J. A., and Moore, A. W. 1995. Generalization in reinforcement learning: Safely approximating the value function. In Tesauro, G.; Touretzky, D. S.; and Leen, T. K., eds., *Advances in Neural Information Processing Systems 7*, 369–376. Cambridge, MA: The MIT Press.

Cassandra, A. R.; Kaelbling, L. P.; and Littman, M. L. 1994. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1023–1028.

Cassandra, A.; Littman, M. L.; and Zhang, N. L. 1997. Incremental Pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–97)*, 54–61. San Francisco, CA: Morgan Kaufmann Publishers.

Chien, S.; Govindjee, A.; Wang, X.; Estlin, T.; and Hill, Jr., R. 1996. Integrating hierarchical task-network and operator-based planning techniques to automate operations of communications antennas. *IEEE Expert* 11(6):9–11.

Chien, S.; DeCoste, D.; Doyle, R.; and Stolorz, P. 1997. Making an impact: artificial intelligence at the Jet Propulsion Laboratory. *AI Magazine* 18(1):103–122.

Chien, S.; Lam, R.; and Vu, Q. 1997. Resource scheduling for a network of communications antennas. In *IEEE Aerospace Conference Proceedings*, 361–373.

Crites, R. H., and Barto, A. G. 1996. Improving elevator performance using reinforcement learning. In Touretzky, D. S.; Mozer, M. C.; and Hasselmo, M. E., eds., *Advances in Neural Information Processing Systems 8.* Cambridge, MA: The MIT Press.

Davis, M.; Logemann, G.; and Loveland, D. 1962. A machine program for theorem proving. *Communications of the ACM* 5:394–397.

Dean, T.; Kaelbling, L. P.; Kirman, J.; and Nicholson, A. 1995. Planning under time constraints in stochastic domains. *Artificial Intelligence* 76(1-2):35–74.

Dearden, R., and Boutilier, C. 1997. Abstraction and approximate decision-theoretic planning. *Artificial Intelligence* 89(1–2):219–283.

Draper, D.; Hanks, S.; and Weld, D. 1994. Probabilistic planning with information gathering and contingent execution. In *Proceedings of the AAAI Spring Symposium on Decision Theoretic Planning*, 76–82.

Drummond, M., and Bresina, J. 1990. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 138–144. Morgan Kaufmann.

Drummond, M.; Bresina, J.; and Swanson, K. 1994. Just-in-case scheduling. In *Proceedings of the 12th National Conference on Artificial Intelligence*, 1098–1104. Seattle, WA: AAAI Press.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208. Reprinted in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate, eds., Morgan Kaufmann, 1990.

Goldman, R. P., and Boddy, M. S. 1994. Epsilon-safe planning. In *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence (UAI94)*, 253–261.

Goldsmith, J.; Littman, M. L.; and Mundhenk, M. 1997. The complexity of plan existence and evaluation in probabilistic domains. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–97)*, 182–189. San Francisco, CA: Morgan Kaufmann Publishers.

Goldsmith, J.; Lusena, C.; and Mundhenk, M. 1996. The complexity of deterministically observable finite-horizon Markov decision processes. Technical Report 268-96, Department of Computer Science, University of Kentucky.

Gordon, G. J. 1995. Stable function approximation in dynamic programming. In Prieditis, A., and Russell, S., eds., *Proceedings of the Twelfth International Conference on Machine Learning*, 261–268. San Francisco, CA: Morgan Kaufmann.

Hansen, E. A. 1994. Cost-effective sensing during plan execution. In *Proceedings of the Twelfth National Conference on Artificial Intelligence.* AAAI Press/The MIT Press. 1029–1035.

Howard, R. A. 1960. *Dynamic Programming and Markov Processes.* Cambridge, Massachusetts: The MIT Press.

Kaelbling, L. P.; Littman, M. L.; and Moore, A. W. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4:237–285.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1194–1201. AAAI Press/The MIT Press.

Kushmerick, N.; Hanks, S.; and Weld, D. S. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76(1-2):239–286.

Littman, M. L.; Cassandra, A. R.; and Kaelbling, L. P. 1995. Learning policies for partially observable environments: Scaling up. In Prieditis, A., and Russell, S., eds., *Proceedings of the Twelfth International Conference on Machine Learning*, 362–370. San Francisco, CA: Morgan Kaufmann.

Littman, M. L.; Cassandra, A. R.; and Kaelbling, L. P. 1996. Efficient dynamic-programming updates in partially observable Markov decision processes. Technical Report CS-95-19, Brown University, Providence, RI.

Littman, M. L. 1997. Probabilistic propositional planning: Representations and complexity. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 748–754. AAAI Press/The MIT Press.

Lovejoy, W. S. 1991. A survey of algorithmic methods for partially observable Markov decision processes. *Annals of Operations Research* 28(1):47–65.

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proceedings of the 9th National Conference on Artificial Intelligence*, 634–639.

Moore, A. W., and Atkeson, C. G. 1993. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning* 13:103–130.

Muscettola, N.; Fry, C.; Rajan, K.; Smith, B.; Chien, S.; Rabideau, G.; and Yan, D. 1997. Onboard planning for New Millennium Deep Space One autonomy. In *IEEE Aerospace Conference Proceedings*, 303–318.

Parr, R., and Russell, S. 1995. Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence.*

Puterman, M. L. 1994. *Markov Decision Processes—Discrete Stochastic Dynamic Programming.* New York, NY: John Wiley & Sons, Inc.

Simmons, R., and Koenig, S. 1995. Probabilistic robot navigation in partially observable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1080–1087.

Singh, S., and Bertsekas, D. 1996. Reinforcement learning for dynamic channel allocation in cellular telephone systems. To appear in *Advances in Neural Information Processing Systems.*

Sutton, R. S. 1988. Learning to predict by the method of temporal differences. *Machine Learning* 3(1):9–44.

Sutton, R. S. 1996. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In Touretzky, D. S.; Mozer, M. C.; and Hasselmo, M. E., eds., *Advances in Neural Information Processing Systems 8.* Cambridge, MA: The MIT Press.

Tash, J., and Russell, S. 1994. Control strategies for a stochastic planner. In *Proceedings of the 12th National Conference on Artificial Intelligence*, 1079–1085.

Tesauro, G. 1995. Temporal difference learning and TD-Gammon. *Communications of the ACM* 58–67.

Tsitsiklis, J. N., and Van Roy, B. 1996a. An analysis of temporal-difference learning with function approximation. Technical Report LIDS-P-2322, Massachusetts Institute of Technology. Available through URL `http://web.mit.edu/bvr/www/td.ps`. To appear in *IEEE Transactions on Automatic Control.*

Tsitsiklis, J. N., and Van Roy, B. 1996b. Feature-based methods for large scale dynamic programming. *Machine Learning* 22(1/2/3):59–94.

Washington, R. 1996. Incremental Markov-model planning. In *Proceedings of TAI-96, Eighth IEEE International Conference on Tools With Artificial Intelligence*, 41–47.

Zhang, W., and Dietterich, T. G. 1995. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intellience*, 1114–1120.

APGEN: A Multi-Mission Semi-Automated Planning Tool

Pierre F. Maldague
Adans Y. Ko
Dennis N. Page
Thomas W. Starbird

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove dr.
Pasadena, CA 91109
maldague@apgen.jpl.nasa.gov

Abstract

This paper discusses a multi-mission planning tool named APGEN, which is currently used by several Flight Projects at JPL. Although APGEN was not intended to function as an autonomous planner, it does meet stringent requirements imposed by current flight project customers. We discuss the nature of these requirements, how they are met in the current implementation of APGEN, and how we are planning to adapt APGEN to the closed-loop environment required by new interplanetary missions.

Introduction and Background

APGEN is a multi-mission planning tool currently in development at JPL. The requirements for APGEN were drawn from the experience of mission planners who were primarily interested in 'traditional' (non-autonomous) methods of commanding spacecraft. To establish the context in which the requirements for APGEN arose, let us start with a brief description of the key concepts of 'plan' and 'sequence'.

In a traditional commanding environment, the final product of the uplink system is a (time-ordered) sequence of time-tagged commands to be executed sequentially by the spacecraft. We refer to this as a 'sequence', and will not be concerned here with (important) details such as the precise representation of each command (ASCII vs. binary, packetized or not, etc.).

A plan, on the other hand, is a collection of activities, usually ordered according to increasing start times. An 'activity' can be defined loosely ('orbit insertion', 'imaging observation') or quite precisely (e. g. as a fully defined maneuver with 32 parameters specifying the timing of all related events, the delta V and change in attitude, etc.) Such a plan is summarized in a graphical timeline, each activity being a horizontal bar stretching from its start time to its end time.

JPL has developed a sequencing tool, SEQ_GEN, that allows uplink personnel to 'wrap' many commands into higher-level 'activities'. The spacecraft can be commanded at the higher, 'activity' level (which is much easier than trying to send the spacecraft individual commands). The process of 'expanding' activities into sequences is made automatic by SEQ_GEN. Without getting into details, the main steps in the 'adaptation' process that customizes SEQ_GEN to a specific project are as follows:

- defining the spacecraft commands
- defining activities and how they expand into commands
- defining the spacecraft model and how commands affect it
- defining the mission rules and implementing them as constraints

SEQ_GEN is currently being used by a variety of missions. From a planning perspective, however, SEQ_GEN is not an ideal tool, because not much can be done until all the adaptation steps have been carried out. SEQ_GEN is best when used for computation and analysis of details at the spacecraft command level; the SEQ_GEN user should really be familiar with the expansion rules as well as the nature of the spacecraft model in order to use it effectively.

Planning personnel felt that there ought to be a tool that allows them to produce 'rough' mission plans well before the details of the spacecraft commands are known. A key requirement is that this tool should *not* require the complex machinery of SEQ_GEN expansion and modeling before it could produce some meaningful results.

This tool was named APGEN, and the main requirements

that it should satisfy are as follows:

- be easy and intuitive to use; in particular, easier to use than pre-existing sequencing tools such as SEQ_GEN and Plan-IT-II
- be able to represent simple resources such as Power and Fuel
- be able to model the effect of activities on resources in a simplified way
- be intuitive to operate
- be able to interface with sequencing tools, in particular SEQ_GEN (details were left unspecified)
- be able to operate in networked fashion, with several users sharing data

Interestingly, at the time APGEN was started, interviews with mission planners showed little interest in spacecraft autonomy, or even in autonomous planning on the ground. The primary goal was to provide mission planners (for the Cassini mission in particular) with a tool that could help a human planner well before the sequencing tools were 'adapted'.

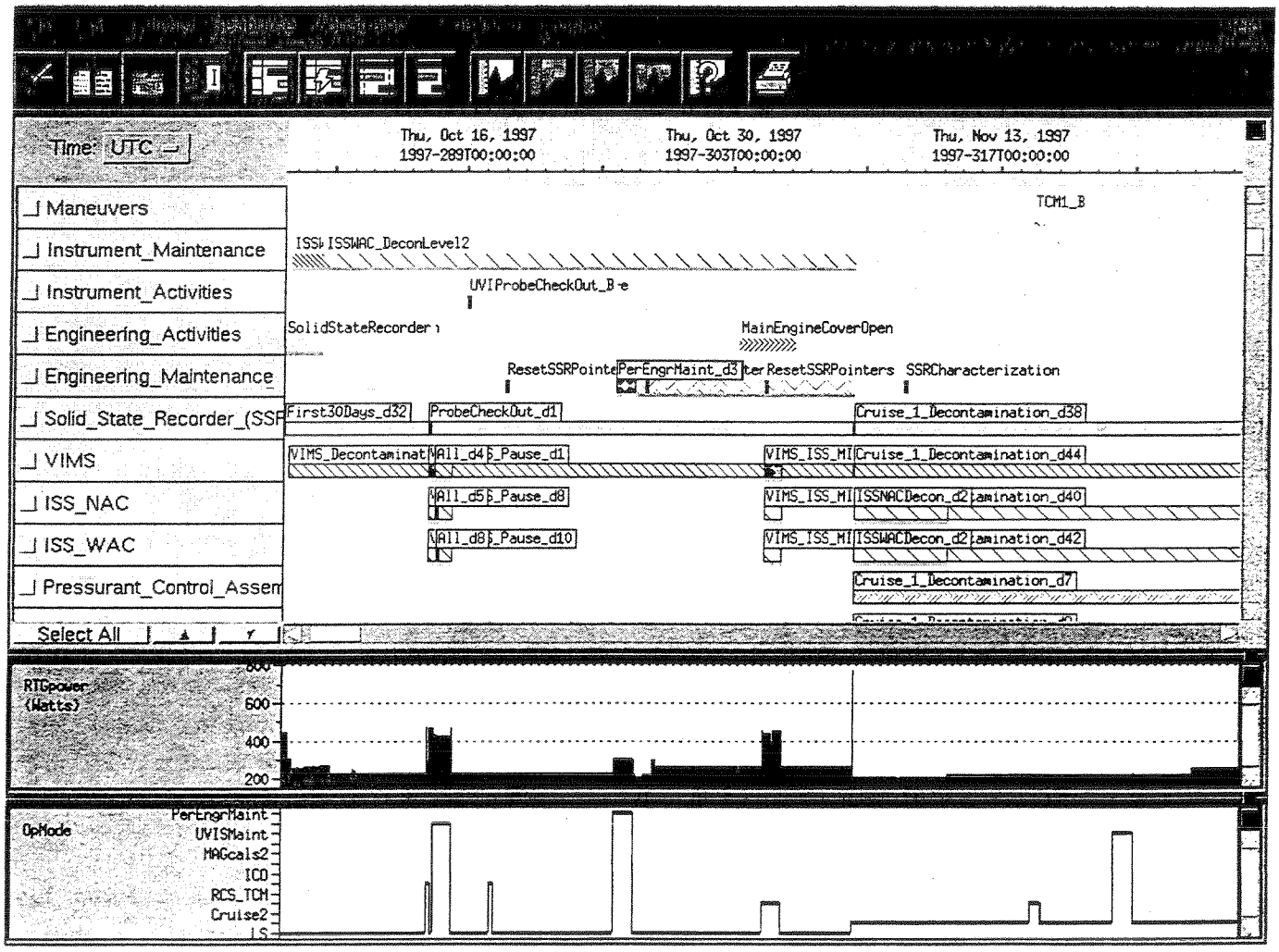


Figure 1: Partial View of the Cassini Inner Cruise Activity Plan as Displayed by APGEN

The Early Days: the Cassini Cruise Plan

APGEN was developed in a succession of engineering versions. As soon as APGEN became able to let an activity 'consume' a resource, Cassini planners started using it to put the Inner Cruise Activity Plan (IACP) together. The ICAP covers the part of the Mission Plan that extends from launch (October, 1997) until the switch to the High-Gain Antenna (February, 2000). The primary target audience for the ICAP is the Mission Planning Virtual Team (MPVT), which consists of 5 Systems Engineers plus representatives from all Science and Engineering Subsystems, or about 100 people in all. The Systems Engineers are in charge of both the adaptation (specifying resources, activity types and how they interact) and the actual plan. Plans are then communicated to remote sites (including several sites in Europe) where APGEN is used to view the plan. See Fig. 1 for a representative display of this plan.

The ICAP plan consists of about 250 activity types; these activity types contain 'usage clauses' that specify how they affect 40 different spacecraft resources. Although APGEN pretty much lived up to the expectations of its users, there were interesting surprises in the particular way the Cassini planners decided to use APGEN capabilities.

The first such surprise has to do with decomposition, which lets users switch between high-level and low-level views of (usually complex) activities. The designers of APGEN envisioned that adapters would use this feature for implementing successive versions of the same activity, each one more refined than the previous one. For example, a crude activity entitled 'orbit insertion', with a simple specification of its average resource use, might later be decomposed into a more detailed 'maneuver' with more realistic parameters, and with a more realistic pattern of resource usage.

In reality, however, Cassini planners ended up using decomposition to define hierarchical links between activities that influence resources simultaneously (a relationship named 'non-exclusive decomposition' in the APGEN jargon). This allows users to represent a complex maneuver as a single activity bar in the timeline, and yet to see the actual, detailed influence of that maneuver on all the resources it impacts as dictated by the sub-activities it contains.

A second surprise is that only about half the resources defined in the ICAP are physical resources that can be used to express planning constraints. The other half is intended to convey information about the plan. For instance, a scientist interested in a specific instrument can focus on the periods of activity of his/her instrument by looking at a resource that specifies the power mode for that instrument ('on', 'off', 'standby' etc.).

Perhaps the biggest surprise for the Cassini team was to realize that their APGEN plans would not automatically convert into a sequence acceptable to their sequencing engine (SEQ_GEN). APGEN has an option for producing files containing 'activity requests' in the proper format for input into SEQ_GEN. However, these files are meaningless unless they match the SEQ_GEN 'adaptation files', which specify the types of parameters of each activity as well as how each activity expands into subactivities and commands. Only after the first few request files were generated by APGEN did it become obvious that coordination between the planning and sequencing teams was essential to obtaining a smooth interface.

New Customers: the Push for More Planning Automation

New potential customers for APGEN include the Space Infra-Red Telescope Facility (SIRTF) and X2000. Both projects are seeking cheaper, more automated ways to conduct space missions. This presents APGEN with two new challenges:

- automatic sequencing of large numbers (thousands) of requests for observations
- modeling of activities whose expansion depends on the state of the spacecraft (conditional expansion)
- more generally, support closed-loop operation as opposed to the open-loop environment typical of ground operations

To meet the first challenge, new features for task scheduling have been incorporated into APGEN. In a nutshell, this allows APGEN to schedule certain activities only when certain conditions are met for a specified length of time. This capability does not make APGEN into a full-fledged 'automatic planner' (it still lacks the capability to refine and optimize proposed schedules). However this incremental approach has the merit of being quite fast and deterministic/predictable. Fig. 2 below shows an eample of this scheduling capability applied to the SIRTF project.

There is one area in which APGEN's 'traditional personality' provides it with an advantage over more modern planning tools: processing speed. Prompted by Cassini users who wanted quick access to their two-year Cruise Plan, APGEN developers had to introduce a number of time-saving features into the design. For example, the original design of

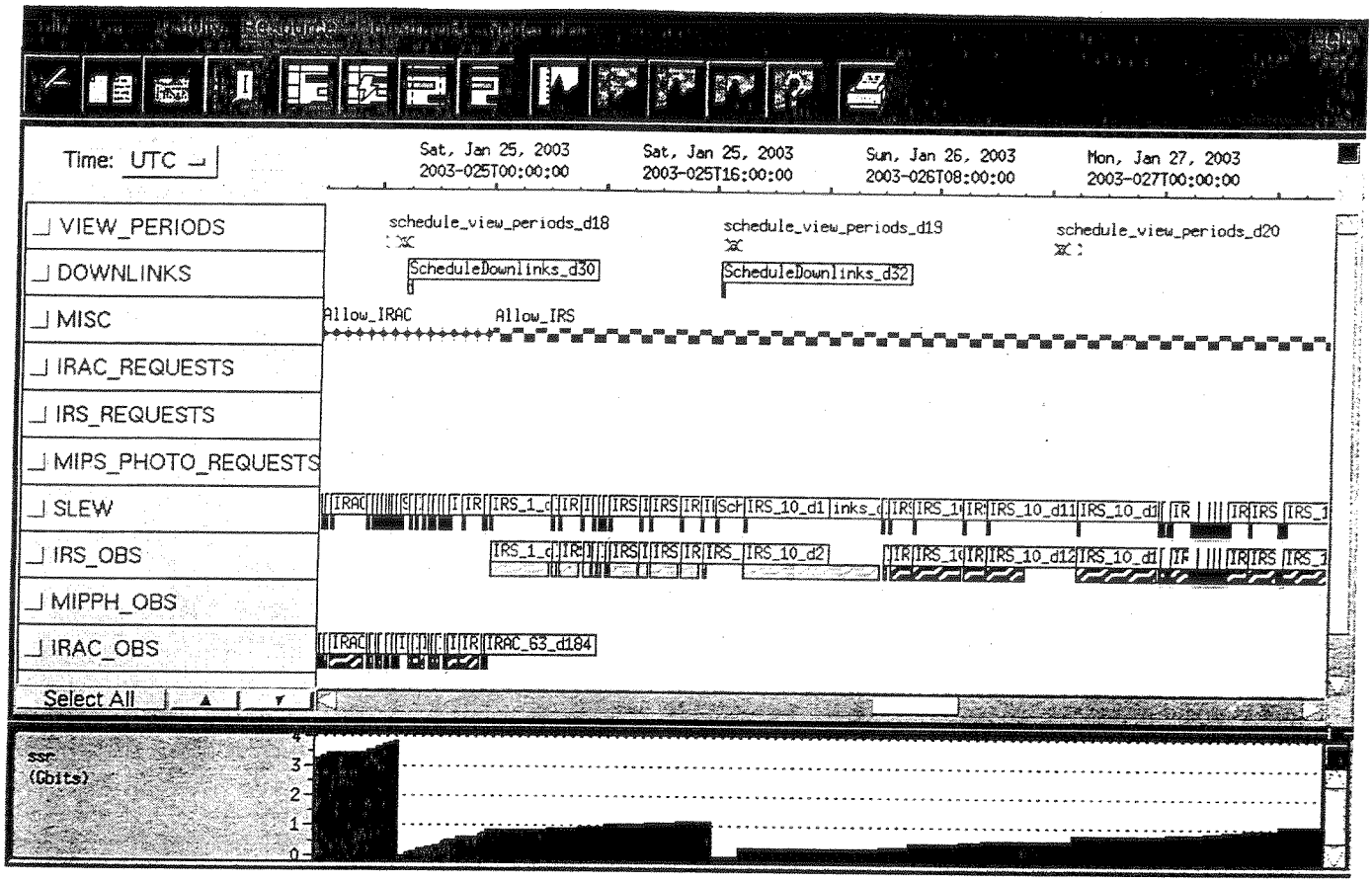


Figure 2: Partial View of a Preliminary Activity Plan as Scheduled by APGEN

APGEN featured an interpreter which was invoked every time the adaptation data had to be consulted. In the new design, the parser saves the parsing trees associated with adaptation data in the form of 'pseudo-code'. When adaptation data need to be consulted, APGEN 'executes' the pseudo-code without having to re-interpret the original data. A second example is provided by the many lists used to store activity types, activity instances, resources, constraints and many of their class members. The original design used a simple linked list scheme, which was adequate for testing purposes but brought the program to its knees when dealing with large numbers of objects. In the current design, the linked lists have been replaced by much more efficient balanced trees. The AVL algorithm (Knuth 1973) is used to insert and delete elements in logarithmic time (i. e., execution time grows as log(N) where N is the size of the list). These two changes resulted in a hundredfold performance improvement.

The second and third challenges are more difficult, because they amount to replacing fixed, time-tagged commands by a 'program' containing conditional (and perhaps iterative) statements. How does one validate such a beast? We don't know.

The Future: ASPEN, SEQ_talk, Autonomy

There are three areas of development planned for the next year. One is to enable APGEN to execute SEQ_GEN and access its results. In this way, as a project's development proceeds to the point of having adapted SEQ_GEN with its detailed models, results of such detailed modeling can be used in APGEN's planning. APGEN would serve as the principal interface to the user, but computations would be shared by the two programs. The programs will be linked by socket communication, leaving them loosely coupled, able to proceed independently with their own developments of other features. Similar linkage, referred to as SEQ_talk, to other existing sequencing and planning tools is being considered. The ability for APGEN to talk to other programs would help to solve one of the surprises encountered by the

Cassini project; computation would be done by the program best suited, but the results would be available to APGEN and its user.

Another area of development is to add access by APGEN to more intelligent planning capability. Use will be made of ASPEN (Automated Scheduling and Planning Engine), which includes a library of automated planning functions, and which is under development at JPL. Again, a loose coupling mechanism based on sockets will be used, enabling parallel development of APGEN and ASPEN.

The third area of development concerns the migration path from ground to flight software. We have recently proposed a concept which we call 'Just-In-Time Planning', in which an APGEN-like planner running on board the spacecraft would handle short-term scheduling tasks that require fast turn-around, such as faults and real-time events. This short-term planner would operate under the direction of a resource allocation manager, which could eventually be extended to a separate, on-board long-term planner.

Acknowledgments

Many of the concepts implemented in APGEN were prototyped in a program called Plan-IT-II, implemented at JPL. Plan-IT-II has a long history of pathfinding experimentation in concepts useful for practical automated planning. APGEN is funded by the Telecommunications and Mission Operations Directorate at the Jet Propulsion Laboratory, operated by the California Institute of Technology under contract by the National Aeronautics and Space Administration.

References

Knuth, Donald E. 1973. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Reading, Mass.: Addison-Wesley.

Planning and Scheduling to Support EOS Science Data Processing

Dan Marinelli
NASA/Goddard Space Flight Center
Greenbelt, MD 20771
dan.marinelli@gsfc.nasa.gov

Abstract

The architecture that has been designed to provide planning and scheduling support for EOS science data processing is described. An operational concept is presented showing how the software interacts to support the production of a very large number of data products which are archived and disseminated to a user community.

Overview

The EOSDIS Core System (ECS) is being developed to produce 260 standard data products which represent information-creation from raw instrument data. Daily, ECS will process 480 gigabytes of this raw instrument data and produce 1.6 terabytes of value-added information. This information is mostly geophysical parameters derived from radiances measured from instruments onboard orbiting spacecraft. Instrument teams provide the knowledge which is coded into science processing software which then executes within the ECS framework to produce information. Several attempts at producing what the instrument teams, or ITs, would call correct information will be needed as the characteristics of the operating instrument become understood and as scientific learning occurs. There is also the desire to share information among instruments because of their complementary nature. Because the ECS will be instantiated at multiple sites, there will be intersite data dependencies that must be managed.

The ECS (EOSDIS Core System) architecture to support execution of science processing software is primarily composed of two subsystems, Planning and Data Processing. These two subsystems are a part of the larger ECS which also provides long term data storage needs, ingest of data from external entities, an interoperability infrastructure, system-wide distributed search and access services, and a graphical user interface for querying and ordering the data. The Planning subsystem provides a human interface to planning the execution of jobs and to planning resource usage. The Data Processing subsystem is responsible for scheduling the jobs on the available resources and keeping operations staff updated on progress. This paper gives an overview of the architecture which is being built to support these functions.

The term "planning", in the domain of ECS, refers to laying out what activities should occur over some timeline so that operations can be efficiently planned and so that the performance of the system can be measured against expectations. Activities related to planning are implemented in the Planning subsystem. "Scheduling" refers to carrying out the plan given a set of resources and dynamic events such as data arrival. Scheduling activities are implemented in the Data Processing subsystem. The two subsystems represent a data driven system in that the plan is a prediction of events that will occur. Processes will only be scheduled once all required data is available. Processes will only execute once they are scheduled and all required physical resources are available.

ECS Architecture

ECS is composed of nine subsystems which interface to meet the demands of ingesting, processing, and distributing EOS data. Two of these will be described here, Planning and Data Processing, but the interfaces to the other subsystems are necessary to accomplish the goals of planning and scheduling. The other subsystems are: Interoperability, Communications, Management, Data Server, Client, Data Management, and Ingest.

The division of the Planning and Data Processing subsystems was influenced by the following factors:

- Follows Client/Server Architecture - Planning acts as the Client, and Data Processing acts as the Server. Planning is responsible for developing a production plan and informing Data Processing which activities must be executed as listed in the production plan.

- Increases Fault Tolerance of the Planning and Data Processing Subsystems - By separating the Planning services from the Data Processing services, an increase in the fault tolerance of the Planning and Data Processing subsystem software is achieved. The failure of the Planning subsystem will not cause an

immediate breakdown in production processing. Data Processing will be able to continue with production processing until the processing queues are depleted. Also, a failure of the Data Processing software or hardware will not cause the Planning subsystem to fail. Although the flow of information between Planning and Data Processing will be interrupted, this should not affect the overall production plan.

- Evolvability of the roles and responsibilities of Planning as ECS becomes operational - The separation of Planning services from Data Processing services allows for different configurations of Planning services and Data Processing services. For example, there may be some special event which would result in the Planning services of a DAAC creating a production plan for itself as well as other DAACs. By separating Planning from Data Processing this becomes an easier problem to resolve.

Planning Subsystem Architecture

The Planning subsystem has three main responsibilities: 1) defining the goals for data production, 2) preparing plans for production, and 3) coordinating the production from those plans.

To support defining production goals, the system uses PGE (Product Generation Executive) profiles. PGEs contain executables and scripts that perform desired transformations on the input science data to produce the output science data. The function of the system is to execute PGEs repetitively with new data as it arrives. The profile for a PGE gives its resource usage requirements to the planning subsystem. Production rules state under which conditions the PGE should execute, e.g., should run each time data comes in, or only a fourth of the time, should it use alternate inputs if the primary doesn't arrive, etc. The human interaction comes in the form of production requests to actually execute the PGEs. These production requests are submitted to a candidate plan which becomes the statement of the operator's goal of what the system should accomplish. The candidate plan is generated 'on top of', or using, a resource plan that describes the availability of the hardware resources. Hardware resources are modeled as CPUs, disks, networks, and platforms which are combinations of CPUs and disks. Resources can be allocated to non-processing activities such as maintenance, test, or training. The description of the resources which exist in the system are provided from a configuration managed database in the Management Subsystem.

The ability to generate candidate plans is important to operations for several reasons. Disconnects in data that may be required to execute some PGEs can be spotted and point to corrective action. Efficiencies can be gained by examining the expected completion times of different plans which gives operations a better feel for what works and what doesn't. Staffing projections can be supported in cases where PGEs require a man-in-the-loop activity for verifying results. Since the ECS executes at several sites called DAACs (Distributed Active Archive Centers), each requiring data from other DAACs, it is important to share the candidate plans by publishing them. Once published, they can be extracted by other Planning subsystems which execute at related DAACs or at the SMC (System Monitoring and Coordination) for a system-wide analysis of the interdependent plans. The planning horizon can go to 30 days.

Once the candidate plans are evaluated, one is chosen for activation. Activation is the process of merging the new plan into the system. Each production request for a PGE causes many DPRs (Data Processing Requests) to be planned and later executed. The previously active plan contains DPRs that match those in the new plan, so these are reconciled and then the new DPRs are added to the list of things to do. As time passes, DPRs execute, and various other events occur in the Data Processing subsystem. The plan does not imply that a DPR must execute at a specific time of day, rather it says what the goal would be if other DPRs executed and the system performed as expected up until that time. The Data Processing subsystem's goal is to execute the DPRs upon receipt of all inputs, and to use it's resources efficiently while doing so. The reality is that DPRs won't all execute to the specifics of their PGE resource profiles and there will be system outages. Should the plan deviate greatly from the expected, or should resources become unused to any large degree, operations staff can replan. A replan is simply the generation and activation of another candidate plan. Another reality is that there may be high priority processing requests that don't get planned, especially in the early mission phases, but also in periods of critical spacecraft activity such as special maneuvers. Requests may be submitted without planning directly into the system and executed. PDPS also supports the concept of 'on-demand' requests. Some resource bandwidth can be set aside in anticipation of these requests during the planning period.

The Planning subsystem is comprised of various utility applications and servers. At its core is the PDPS Database which serves as the central repository of all information used by both the Planning and Data Processing subsystems. A high-level software architecture is shown in Figure 1.

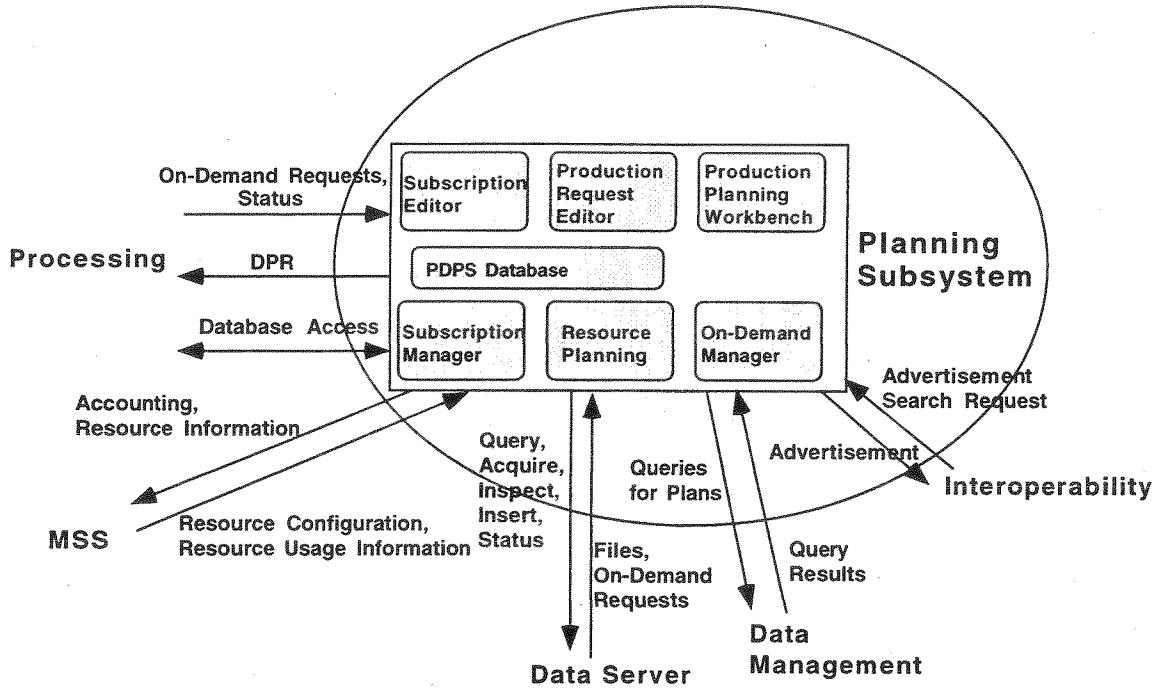


Figure 1 - Planning Subsystem High Level Software Architecture

The utilities, applications that execute in support of human interaction with the system, are:

- Production Request Editor - allows the operator to submit production requests for the data to be produced,
- Production Planning Workbench - used to prepare a plan for the production at a site and forecast the start and completion times of the activities in the plan,
- Planning Subscription Editor - provides the capability of reviewing subscriptions to data that are required for production, and,
- Resource Planning Workbench - allows operations to allocate and plan the use of the hardware resources.

The servers, which execute continuously so that work keeps flowing, are:

- Subscription Manager - waits for and then acts upon subscription notifications for data that arrives into the ECS,
- On-Demand manager - manages the receipt of on-demand requests, i.e., those that aren't planned for except in the general sense of reserving a set of resources for their execution, and,
- PDPS Database Server - provides secure, fault tolerant, concurrent access to system data.

The hardware configuration for PDPS at a large site is shown in Figure 2. The allocation of software to hardware is shown in Figure 3. The drivers for the Planning half of this hardware selection are:

- the PDPS database for storing persistent data
- backup and recovery for the PDPS database, and,
- growth.

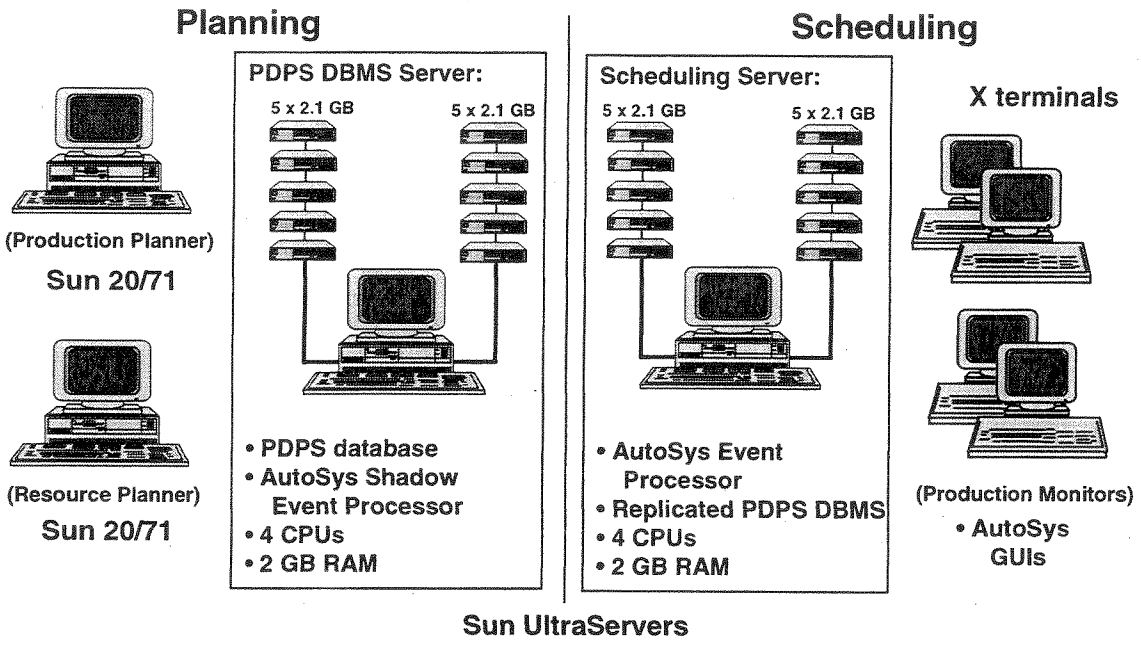


Figure 2 - PDPS Hardware Configuration (sans Science Processors)

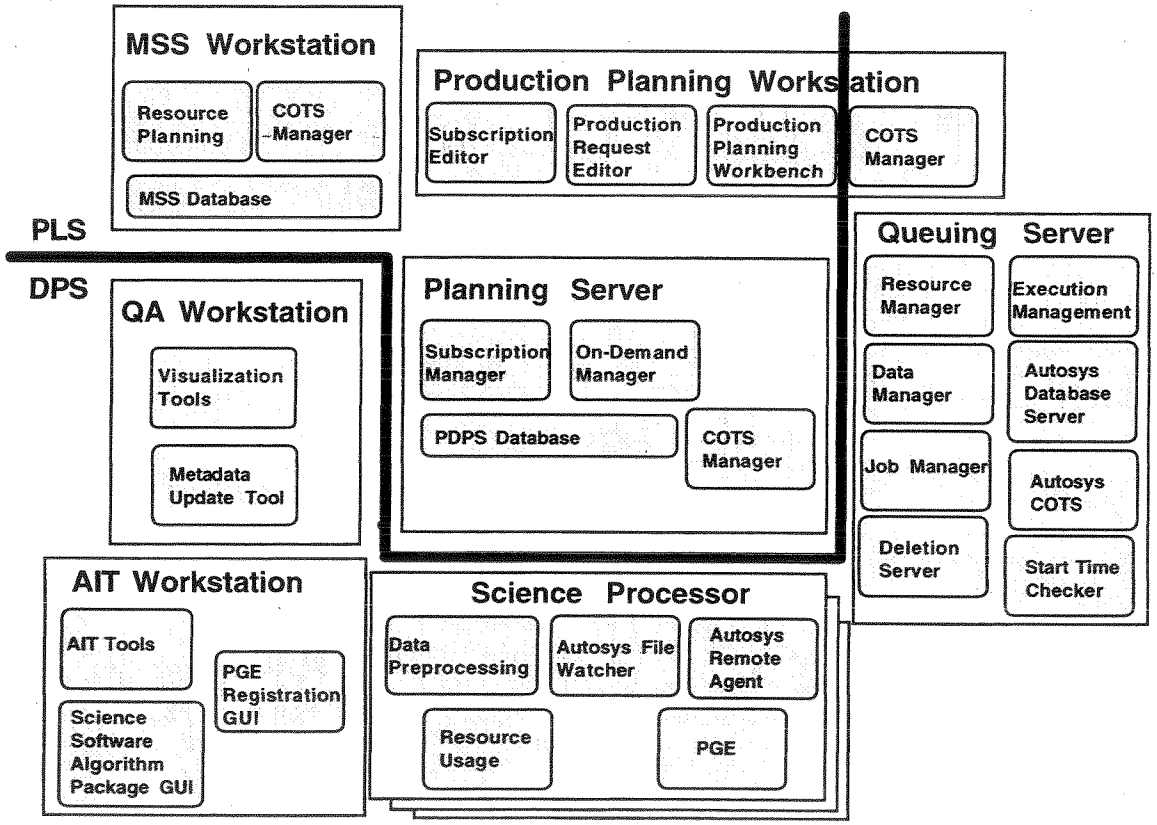


Figure 3 - PDPS Allocation of Software to Hardware

Operations staff must be able to generate a candidate plan in a 'reasonable' amount of time. Since candidate plan generation should occur at most once per day, with once per week the nominal scenario in a steady-state system, two hours is thought to be a reasonable time. For the largest site, there will be about 32,000 jobs per day. The database transactions are the limiting factor in the plan generation. For 30 day plan generation there will be an estimated 9 million DBMS transactions. The 4-CPU Sun UltraServer transactions per second rating is 1360, thus it would take about 110 minutes to generate the 30 day plan.

This configuration provides PDPS database replication to meet the backup/recovery requirement with adequate performance to meet the plan generation times expected. Growth is possible with the addition of CPUs and disk with no software changes. For smaller sites there is a slightly different configuration that meets lower transaction per second requirements.

Data Processing Subsystem Architecture

The Data Processing subsystem must perform two main functions: 1) support science algorithm development, and 2) support execution of science software while making efficient use of the science data processing resources.

Science algorithm development is supported with the Science Data Processing Toolkit, "the Toolkit", and with a set of tools for integration of the algorithm software, "the AIT (Algorithm Integration and Test) environment". The Toolkit was delivered early so that science software development teams could proceed on their development path independently of ECS. It has multiple language bindings and runs on many platforms. The goal is to eliminate redundant development activities among the development teams and to streamline the science software integration process. The library provides interfaces that give access to ECS products, product metadata, ancillary data and processing parameter information. The AIT environment is a set of tools that facilitate the integration of science software into the ECS DAAC environment.

The Data Processing subsystem provides a means of monitoring and managing the queued generation of scientific data products. DAAC Science Data Processing is not a real-time, automatically invoked, single data stream, uniform process such as spacecraft telemetry processing. Science data must be collected, stored, prepared, and "batched" for generation of data products. Processing occurs post (satellite) pass and must be planned on the basis of data quality and availability, and the availability of appropriate system resources.

The planning for the generation of data products is performed by the Planning Subsystem. When the candidate plan is activated, the set of DPRs that is to run in the earlier phases of the plan are passed to Data Processing via the PDPS database. The Autosys COTS product, which performs all the scheduling functionality, determines from the list of DPRs which ones can execute immediately. This is done by checking a data availability flag. The data availability flag is used to signify that the data exists within ECS, not necessarily on the Data Processing local disk resource. Those DPRs for which data are available are metered to the processing resources based on resource availability (CPU, memory, and disk space). The ones which don't have their data are considered by Autosys as 'on-hold'.

Planning has the ability to query the Autosys COTS to determine status of DPRs that have been submitted. As data arrives into ECS, Planning is made aware via its subscription manager and can signal to Data Processing through the PDPS database that pending DPRs can execute. Processing must perform several actions in support of executing the DPR. Each DPR is executed using 4 Autosys jobs which perform the functions of preparation, execution, and cleanup. Each of these 3 functions is wrapped into a larger job so that GUIs can be more easily used to identify activities which simplifies monitoring. Once Data Processing determines that a DPR can execute, i.e., all resources required are available and all data exists within ECS, the Autosys COTS is given the responsibility of executing the jobs. The first job that is 'kicked-off' is the entire job box which is the wrapper for the jobs that do the work which executes the DPR. This results in the kick-off of the preparation job. The responsibilities of the preparation job are to allocate disk resources, load data onto those resources that are needed by the DPR, and load the control information into the Process Control File that is used by the individual PGE which is to execute. Completion of this job results in the kick-off of the execution job. This job executes the PGE. Upon completion, the cleanup job is executed. This job destages data to the Data Server Subsystem for long-term archive, stores production history for the PGE in the Data Server, and deallocates resources. Should the data that has just been created be required by a downstream DPR, that data will be left on the local disk instead of deleted.

As the jobs are submitted to Data Processing and as they execute, the Autosys GUI can be used to see the progression of execution from 'on-hold' to 'executing' to 'complete'. Color coding is used to indicate problem areas and to give indicators of what is happening.

The high-level software architecture for Data Processing is shown in Figure 4. The AIT portion serves as the

'definition' stage of processing. I.e., the PGEs which are to execute are integrated into the system with this functionality. All information which the PDPS needs to support execution of the PGEs is provided at this stage. CPU, disk, and memory requirements are logged into the PDPS database as well as the structure of the Program Configuration File that is used to execute the science software.

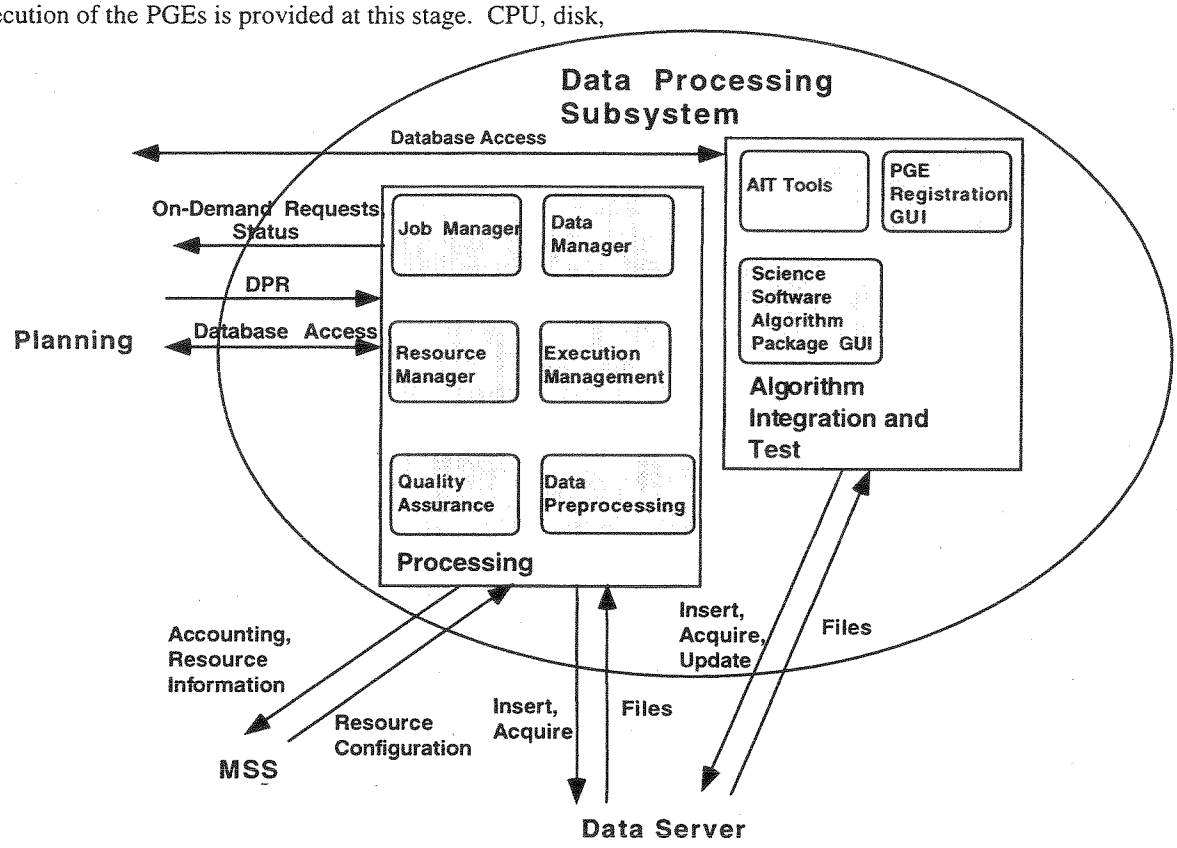


Figure 4 - Data Processing Subsystem High Level Software Architecture

The servers which execute in PDPS are:

- Deletion Server - controls when to delete files that are kept for short periods of time to support processing,
- Job Management Server - provides the interface between Data Processing and the Autosys COTS, encapsulating it,
- StartTimeChecker - continually checks to see if there are any jobs that haven't executed beyond what is considered a reasonable time, and,
- Compression Server - performs compression on data that is to be archived at remote sites.

The remainder of the components, not all of which are discussed in this paper are:

- Resource Management - manages hardware during production,
- Data Management - handles data staging from the Data Server, and data destaging to the Data Server,
- PGE Execution Management - performs pre- and post- PGE execution management functions including generation of the Process Control Files, and bundling of log files,
- Autosys COTS - used to run and monitor jobs on the science processing hardware,
- Quality Assurance Monitor - simple interface allowing DAAC operators to perform QA functions on output data, and,
- Data Preprocessing - a set of PGEs that supports conversion of orbit and attitude data into formats usable by the SDP Toolkit.

The drivers for the Data Processing hardware selection are job throughput and growth. At the largest DAAC, approximately 32,000 jobs per day must flow through the system. The current design will allow for about 190,000 jobs per day and with just the addition of hardware includ-

ing extra RAM and spinning disk, the design will allow for 560,000 jobs per day.

Summary

The ECS Planning and Data Processing subsystems, with support from the rest of ECS and the science software delivered by the EOS Instrument Teams, are designed to provide the capabilities needed to manage science data processing in a distributed environment. Integration of science processing software is facilitated by the SDP Toolkit and the Algorithm Integration and Test tools. PGEs are integrated and their resource usage profiles are defined to the system. The ability to adjust which data to process is provided with the use of production rules. Resource usage is optimized by the Planning subsystem which enables operations staff to plan routine production, reprocessing, and on-demand activities and to schedule maintenance activities. The Data Processing subsystem is responsible for effectively managing all resources in real-time. It manages the data into and out of the subsystem hardware, and keeps track of whether it's needed in the near future so that network traffic to the Data Server is optimized.

More information on EOSDIS and the ECS can be found at: http://ecsinfo.hitc.com.

Acknowledgments

The work that this paper describes is being done by the people at Hughes Information Technology Systems in Landover, MD. Special thanks to Will Knauss, PDPS Manager, for providing timely information and proofreading of this work.

References

Bopf, M. PDPS Introduction, 706-CD-003-001 Day 1. Presentation for ECS Critical Design Review for Release B. Prepared under Contract NAS5-60000 by Hughes Information Technology Systems. April, 1996.

Fox, S.; Prasad, N.; and Szczur, M. NASA's Earth Observing System Data and Information System (EOSDIS): An Integrated System for Processing, Archiving, and Disseminating High-Volume Earth Science Imagery and Associated Products, 215-TP-001-001. July, 1996.

Release B SDPS Data Processing Subsystem Design Specification for the ECS Project, 305-CD-027-002. Prepared under Contract NAS5-60000 by Hughes Information Technology Systems. March, 1996.

Release B SDPS Planning Subsystem Design Specification for the ECS Project, 305-CD-026-002. Prepared under Contract NAS5-60000 by Hughes Information Technology Systems. March, 1996.

Roth, G. Release B Planning and Scheduling Hardware Selection, 706-CD-003-001 Day 6. Presentation for ECS Critical Design Review for Release B. Prepared under Contract NAS5-60000 by Hughes Information Technology Systems. April, 1996.

Modeling Constraints for Scheduling Problems

Mary Beth McMahon

The Boeing Company
13100 Space Center Blvd
Houston, TX 77059

Abstract

Commercial off-the-shelf products are often selected as a solution when faced with a new scheduling problem. These products perform poorly on finite capacity scheduling problems because they assume infinite resources and they do not model some of the harder constraints. This paper deals with our experience in deploying a finite capacity scheduler to several working environments. It describes the constraints that we encountered that caused us to modify the finite capacity scheduler in order to meet real world demands.

Introduction

When using some of the popular commercially available tools for scheduling, the current state-of-practice is based on constraints such as temporal constraints, precedence relationships and simple resource requirements. Commercial tools are often chosen because they are marketed as a pre-packaged solution and the advertisements show the public nice reports and friendly user interfaces that make it look easy to use the product to solve their problems. At first, the product seems to produce satisfactory results for the user. But as the user becomes more experienced with the product and the intricacies of the scheduling problem, the user runs into real-world constraints that cannot be modeled. This has been our experience for the past eight years when working with customers that were trying to improve their scheduling capabilities.

Some of the working environments in which we deployed a finite capacity scheduler included a testing lab where people load software and run simulations, a test and integration lab where people test hardware and software against different configurations, the SpaceHab Module, and the factory floor where people build aircraft. In adapting our software to meet their scheduling needs we had to model and schedule against some constraints that we had not thought about beforehand. Many of these constraints are generic enough that they are found in a number of domains in the space industry including assembly, mission planning, and mission operations.

The Underlying Scheduling Paradigm

Before discussing the constraints that need to be modeled, it is worthwhile to make sure there is a common understanding of the underlying scheduling paradigm that our finite capacity scheduler uses.

Each piece of work that needs to be done is modeled as a task. Each task knows about its own constraints: its earliest start time, the resources it needs, etc. The number and types of constraints are dependent on the application.

A Precedence Diagram (PD) can be used to model the precedence relationships between tasks. A PD is a diagram with boxes for tasks and arrows depicting precedence.

We define each resource, along with its initial availability, net usage, and list of attributes. An attribute is any characteristic by which a resource may be called upon, e.g. the skills associated with the resource (skill matrix). As tasks are scheduled and unscheduled, the net usage profile gets updated.

When a task gets scheduled it occupies a portion of the timeline and reserves the resources it requires. Subsequent tasks are scheduled around the tasks that are already scheduled. In our paradigm, scheduling a new task will not cause any already scheduled task to be effected in any way. Some schedulers shift tasks around to make room for a new task. Whether or not to perturb the existing schedule when adding tasks is an important consideration when selecting a scheduling tool.

In most of our scheduling problems, the goal of the scheduler is to come up with the shortest possible schedule while adhering to all specified constraints. There are numerous other optimization schemes, however for purposes of this paper minimizing makespan is a sufficient goal.

The Constraints

The following section describes the various constraints and behaviors that we needed to learn to model in order to tailor

our system to handle real-world problems. The constraints included in this section are ones that most commercial schedulers fail to recognize. They are also constraints that can be found when scheduling space-related applications.

Constraint #1: Minimum/Maximum Duration

The duration of the task is a best guess at the length of time it will actually take to perform the work. The duration of a task may be more complex than just a singe number representing days and hours. In some cases the duration is flexible – usually there is a minimum duration and a maximum duration. The duration of the task should fall between the two. In the case of Boeing's Avionics Integration Lab, the scheduler gives each user the minimum duration requested and then backfills each task trying to stretch it to its maximum duration. This gives the user "as much time as possible" up to their maximum duration in the lab, while still ensuring that everyone gets in.

Constraint #2: Interruptible Tasks

Often tasks longer than a day must be interruptible or splittable. When splitting a task, rules about breaking it up must be captured. For example, if you do not specify the minimum time slice, or the shortest amount of time required to do productive work, then the scheduler may spread an hour long task out into 60 one-minute jobs. When splitting a task you may want to find out

- the minimum time slice,
- the maximum time slice,
- the minimum time between slices,
- the maximum time between slices, and
- the maximum time in which to accomplish the task (time space from start to finish including breaks).

Constraint #3: Tracking Tasks

Once the schedule goes into production, the tasks must be updated as work gets completed and a new schedule must be published. Often tracking is done by interfacing the scheduling system to existing systems, such as a timekeeping system. In some cases email is used as the interface to the user in order to collect requests and amount of work completed.

In any case, the kinds of information that needs to be tracked is:

- the planned completion times,
- the actual completion times,
- the amount of work completed,
- the date/time a task is delayed until,
- the reason a task is delayed,
- the cancellation of a task and the reason, and
- the addition of new tasks.

Sometimes work is completed "out of sequence", that is, a task is completed before its predecessors. Although in theory this is not supposed to happen, in the real world it does and the scheduler needs to be able to respond appropriately. Sometimes once the task is completed, it allows its successors to be started. Other times the task's successors must still wait until its predecessors are complete. Sometimes the task completed out of sequence needs to be revisited after its predecessors are complete, usually for some minor work.

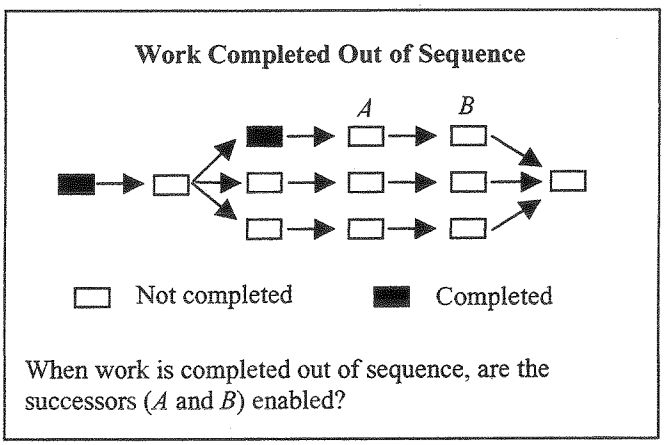


Figure 1. Work Completed Out of Sequence.

Constraint #4: Precedences and Preferences

The first step in deploying a scheduling system is to collect the data that needs to be scheduled. This includes all of the tasks, the resources, and the constraints. When collecting data it is often confusing whether the user is describing a constraint (a real requirement) or just a preference. A scheduler needs to distinguish between the two and model them both.

For example, when users create PDs showing the tasks to be done and their precedence relationships, we often hear that a group of tasks are linked by precedence because that is the order in which they perform the tasks. Even though there is not a requirement for doing them in that order, they assume the link because that is how it has always been done. Often there are good reasons to do the tasks in a particular order, but by modeling these as precedence relationships, we are unnecessarily introducing constraints and that will limit the solution space. However, if we don't model the order, we are losing valuable information - heuristics that a person has discovered about a good order for the tasks.

Modeling preferences and precedences introduces limiting assumptions to the scheduling engine which may compromise it's ability to come up with a better schedule than what is currently being worked. In extreme cases the user may dictate the order of all of the tasks, in which case there is only one solution, and the scheduling engine has lost all power to search for better solutions. In fact, the

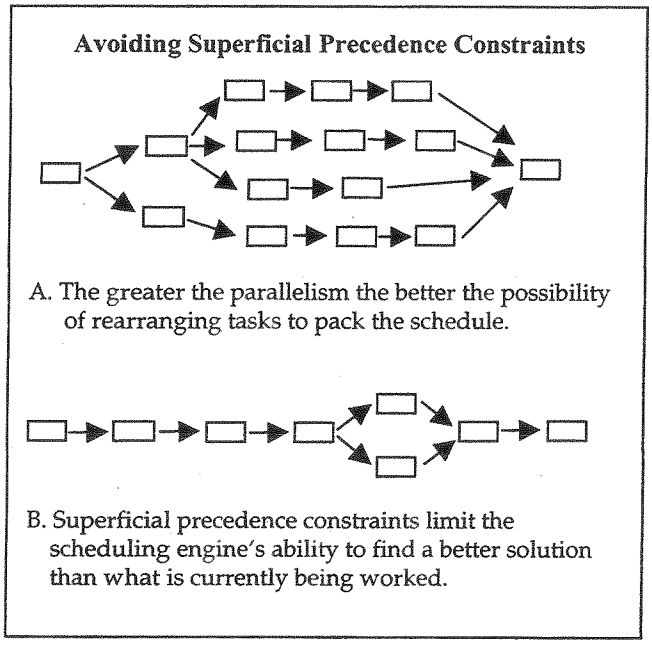


Figure 2. Avoiding Superficial Precedence Constraints

more parallel the tasks are, the greater the size of the solution space.

It is important to model both the real precedence requirements as precedence constraints and the preferred order as preferences. A scheduling engine may violate preferences in order to achieve a better value for an optimization function, but it may never violate precedence constraints.

Constraint #5: People Policies

One of the hardest things to model in scheduling is the subjectivity that goes into building a schedule where people are one of the driving resources. Not only are there union regulations governing working hours/overtime, but there are also "people policies" that managers employ to ensure that the schedule is humane.

In the case of the Systems Engineering Simulation Facility at Johnson Space Center, the workweek is around-the-clock, broken up into 4-hour "sessions." Their people policies include:

- A person can only work up to two sessions in one day and those sessions must be consecutive.
- Each person should have at least an eight hour break between non-consecutive scheduled sessions.
- If a person works more than one third shift then the third shift sessions should be either consecutive or at least two days apart.

In the Integration Lab in St Louis which is also scheduled 24 hours/day, 7 days/week, the people policies include:

- The maximum hours one can work in a "day" is 10 hours, but a day does not have to align itself with normal day to day time boundaries, (e.g. at midnight a new day starts) instead a day is defined by the times the user is assigned work.
- The minimum time between the completion of one day's activities and the start of a new day is twelve hours, eight of which must be on second shift.
- If the user requires specific hours, schedule them within the required hours or not at all. If the user does not require specific hours, but has preferences, let the user specify two sets of preferences. Attempt to schedule the task within the first set of preferences, if that fails attempt to schedule within the union of the first and second set of preferences.

Constraint #6: Preferred Resources

Another time users indicate preferences is when they describe the resources needed to complete a task. If there are multiple sets of equipment that can be used to perform the task, the user often has preferences on which set to use. The scheduling engine must 1) allow the user to specify alternate resources and 2) handle user preferences when selecting resources if applicable.

For example, when working on the assembly of aircraft, there are different categories of skilled laborers including a category called "ama" or "all around machinist/assembler". This person can do the work of a sheet metalist, an electrician, or a mechanic. In most tasks this person is an alternate resource that can be called upon to do the task. However, when scheduling we try to get the specialist first.

Constraint #7: Effect of Resources on Duration

The duration of the task often varies depending on the selection of resources. For example, one machine may mill a part in half the time it takes another machine to mill the part. Different skilled laborers perform the same job at different rates of time. In specifying alternate resources you may also need to specify how each resource effects the duration of the task.

Constraint #8: Limits of Physical Space

Often tasks are limited by the physical space surrounding the areas where the tasks are to be performed. For example, although there are more than enough resources at a given time to perform several tasks inside the cockpit of an F18, the space limits the number of workers to one or two at a time.

Similarly space is a consideration during the flight control play check. A person sits in the cockpit to run the tests. The areas around the aft portion of the wings, the rudders, and the stabilizers must be clear because there are 3000 lbs. of hydraulic pressure in the lines. If there were a single

pinprick in the lines it could seriously injure a person if they happened to be working in any of those areas. So it is imperative to consider the space a task occupies when it is being performed.

It is very complex to model space in its true 3D form. Often a 2D representation or a mapping function can be used to adequately solve the problem. In our case we use "zones". The space around the aircraft is divided into predefined zones and each task can occupy one or more of the zones.

On the shop floors at the Cape where the space shuttles are refurbished, large pieces of equipment need to be moved around to prepare the area for work. If work in progress blocks a passageway and prevents a piece of equipment from being moved to the appropriate place, the tasks using this piece of equipment are delayed. So even though the equipment is available, the physical limitation of space prevents the equipment from being moved and the tasks from being performed. Modeling the geometry of the shop floor becomes an important factor when scheduling this case.

Affecting the Availability of Resources

We have run into three separate occasions when tasks have affected the availability of resources at times *other* than when they were scheduled. Traditionally, a task uses a resource during the time it is scheduled and returns it upon completion. The following examples illustrate instances that deviate from this behavior.

Production and Consumption of Resources

The completion of a task may result in the production of a resource used by a later task. Similarly, a task may consume a resource so that it is no longer available. This is the case in the missile systems area of Boeing. When manufacturing missiles, several tasks build subassemblies. Subsequent tasks use these subassemblies to build more complex subassemblies, and so forth. This building up of subassemblies and their use by subsequent tasks needs to be modeled. One way to do this is to model each type of subassembly as a resource. A task that creates a subassembly "produces" it. While a task that permanently uses a subassembly "consumes" it. The notion of the production and consumption of resources is a useful idea that is prevalent throughout space-related applications - especially in confined environments such as the space station, the shuttle, or SpaceHab, where resources, such as power, are used up and replenished.

The Same Resource for Multiple Tasks

Another occasion where resource availability is effected outside a single task is in the case of airplane assembly. In building large components of the aircraft, pieces are placed onto a jig where they can be riveted, sanded, and soldered. A number of different tasks can be performed while the pieces are attached to the jig. However, these tasks do not need to be performed in any specific order, nor do they have to be performed one right after the other. On many occasions, the labor will start some jig tasks, work on other tasks and then come back to the jig. Note that the jig is not available once the parts are mounted, but other resources such as equipment and labor used by the jig tasks are available to perform other tasks. Since there are a limited number of jigs, they are a highly constrained resource. Therefore the jig must be modeled as a resource that is unavailable once the parts are mounted, and available again once the jobs requiring the jig are complete. So the notion of several tasks securing a single resource needs to be modeled.

Intermittent Resource Changes

Lastly, there are instances when outages, malfunctions, and unplanned maintenance effect the availability of resources. When a resource becomes unavailable due to an unforeseen event, then the effect must be propagated throughout the schedule. A case where this has a significant impact is when aircraft assembly operations in St Louis change from two-shift to three-shift operations and vice-versa due to marketing or management directives. Sometimes this change is in effect for just a few weeks and sometimes it is in effect indefinitely. This kind of impact generally causes all tasks to be rescheduled in order to determine the effect on the schedule. The ability to update resource availability for specified periods of time needs to be modeled.

Overriding Constraints

Once we were able to model the constraints mentioned above, there were times when the manager wanted to override constraints. This is because the manager knew that work-arounds could be found - either people could share equipment, or they could put extra effort into a task to complete it despite the constraints. Sometimes the manager was willing to put up with an infeasible schedule to get a high priority job done, knowing that the conflicts could be dealt with sometime in the future

Almost every customer we have worked with has asked for this type of control. They want the ability to override certain constraints or schedule tasks at a given time, despite what the scheduling engine computes. They want feedback telling how the constraints are being violated because of their decisions. They often want to know the impact on the schedule to get ideas for how to work around the infeasible schedule and get back on track.

A scheduler should be able to enforce or relax constraints at the user's request. It should provide them with informative statistics about the goodness of the schedule and its validity or lack thereof when constraints are relaxed.

Conclusion

Commercial schedulers continue to be popular because they are readily available, easy to learn, and provide some very basic scheduling capabilities However, if the scheduling problem contains some of the advanced constraints described in this paper, there is no alternative but to search for a custom solution. Space operations are fraught with unusual scheduling constraints due to its highly constrained nature, its hazardous operations, and its lack of flexibility. Commercial schedulers which focus on project scheduling provide poor solutions for the types of problems we have encountered where the schedules need to reflect these more advanced constraints.

Evolution of the Spike Planning and Scheduling System

Glenn Miller, Larry Kramer, Peg Stanley, Tony Krueger, and Mark Giuliano
Space Telescope Science Institute (STScI)
3700 San Martin Dr.
Baltimore, MD 21218
{miller, kramer, pstanley, krueger, giuliano}@stsci.edu

Abstract

Spike is a general framework for planning and scheduling that was developed by the Space Telescope Science Institute for NASAUs Hubble Space Telescope. This paper describes how Spike has been adapted to different missions and how the overall system has evolved with experience. Major architectural concepts which have emerged include the development of the Rmultistart stochastic repairS constraint satisfaction problem-based scheduler from an initial neural network approach and the development of planning via Rplan windowsS to provide stable plans in an unstable environment. In addition to over seven years of operational use for HST, Spike has been used for several other space- and ground-based observatories, including EUVE, ASCA and XTE and will be used in FUSE, AXAF and the ground-based European Very Large Telescope. This paper also illustrates how Spike is adapted to new missions.

On Board Planning for Autonomous Spacecraft

Nicola Muscettola
Chuck Fry
Kanna Rajan

Computational Sciences Division
NASA Ames Research Center
Moffet Field, CA 94035
{mus, chucko,kanna}@ptolemy.arc.nasa.gov

Ben Smith
Steve Chien
Gregg Rabideau
David Yan

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, M/S 525-3660
Pasadena, CA 91109-8099
{smith, chien, rabideau, yan}@aig.jpl.nasa.gov

Abstract

The Deep Space One (DS1) mission, scheduled to fly in 1998, will be the first spacecraft to feature an on-board planner. The planner is part of an artificial intelligence based control architecture that comprises a planner/scheduler, a plan execution engine, and a model-based fault diagnosis and reconfiguration engine. This autonomy architecture reduces mission costs and increases mission quality by enabling high-level commanding, robust fault responses, and opportunistic responses to serendipitous events. This paper describes the on-board planning and scheduling component of the DS1 autonomy architecture.

Introduction

The first mission of the New Millennium program (NMP)—Deep Space One (DS1), to launch in 1998—will feature an experimental on-board autonomy software system, the Remote Agent (RA). RA is an artificial intelligence based control system derived from the NewMaap technology demonstration [1]. RA has three components: the Executive (EXEC) [2], the Planner/Scheduler (PS), and the Mode Identification and Recovery engine (MIR) [3]. In this paper we describe PS.

RA is crucial for the future of space exploration because it enables reduction of mission costs and increases mission quality in several ways; some of these improvements are specific to RA's use of an on-board PS. First, because of its hierarchical architecture and its ability to generate plans on-board, RA enables commanding the spacecraft with fewer, high-level commands. Second, RA autonomously responds and recovers from failures that would normally require costly and time consuming ground intervention. PS supports failure response by looking in the future and deliberating about mission goal trade-off and global interactions: the rest of RA handles localized real-time failures that require quick reactions. Finally, on-board planning enables taking advantage of fortuitous events, such as better than expected resource consumption or serendipitous science discoveries (e.g., volcanoes on Io). This opens up fundamentally new and exciting unmanned space exploration missions where round-trip light time does not permit joy-sticking a spacecraft from Earth.

In this paper first we quickly describe the DS1 mission . Then we introduce the concept of high level commanding enabled by RA, contrast it to traditional sequencing, and highlight the role played by PS. Finally we describe more in detail the features and capabilities of the on-board PS.

The DS1 Mission

Spacecraft used by NMP missions are relatively inexpensive (e.g. the DS1 Mission is capped at $138.5 million) and must serve the primary objective of validating new technologies in flight; doing planetary science is an additional objective but one that also has the effect of stress-testing the technologies. This ordering of priorities allows the development and validation of technologies that would not pass the stringent reliability requirements imposed by typical planetary science missions. If the new technologies prove worthy on an NMP flight, then they will be used on future science missions.

The nominal DS1 mission is to launch in July 1998, fly by Asteroid 3352 McAuliffe in January 1999 taking a series of images, and then fly by Comet West-Kohoutek-Ikemura in June 2000. There will be comparatively infrequent ground communication coverage throughout the mission, only one Deep Space Network pass every two weeks. DS1 will carry aboard several new

technologies. During cruise preceding the first encounter each new technology will go through validation experiments. The RA is one of these technologies. Others include the on-board optical navigator (NAV), the ion-propulsion engine (IPS), and the Miniature Integrated Camera Spectrometer (MICAS).

RA is an experimental spacecraft control system. It consists of three components: the Planner/Scheduler (PS), the Executive (EXEC), and the Mode Identification and Recovery system (MIR). PS receives a set of high-level mission goals from an on-board *mission profile* and generates a *plan*—a set of synchronized procedures. Once executed, these commands will achieve the mission goals without violating resource, temporal, or safety constraints. The EXEC takes each plan procedure, decomposes it into low-level real-time commands and ensures the correct dispatching of these commands. MIR monitors device responses to commands, identifies faulty components, and suggests recovery actions to EXEC.

High Level Commanding

The RA architecture enables a new approach to spacecraft commanding, *high-level commanding*. PS is the primary module through which high level commanding happens. In this approach, ground interacts with a spacecraft through abstract directives or *goals* instead of detailed streams of instructions. The responsibilities of PS are: (1) to select among the proposed goals those to be achieved at any point in time; (2) to compromise between the level of achievement of the selected goals, and (3) to expand the *procedures* needed to achieve the goals. PS ensures the satisfaction of various synchronization constraints among procedures and resolves resource conflicts. The set of expanded procedures and constraints among them constitutes a *plan*.

In contrast, in the traditional approach a spacecraft is commanded with a sequence of time-tagged commands to the real-time device drivers. Such a sequence could be highly optimized to "squeeze" as much performance as possible out of the spacecraft. However, temporal and resource constraints and fault protection goals also have to be ensured at an extremely detailed level. The consequence is that developing a sequence is a very exacting and time consuming process, often requiring months of manual labor. Once generated, a sequence is very difficult to modify.

Three are the primary benefits of high-level commanding when compared to traditional sequencing: *modularity, execution flexibility* and *robustness*.

Modularity: a PS plan makes very explicit the hierarchical decomposition of responsibilities between the different flight software components. Each plan procedure is expanded by EXEC into fairly complex sequences of real-time commands on the basis of actual execution conditions. Since the plan already resolves synchronization and resource allocation constraints among procedures, this expansion is highly localized and, therefore, greatly simplified. Extensive validation of these small sequences is much simpler than the validation of those generated in the traditional approach.

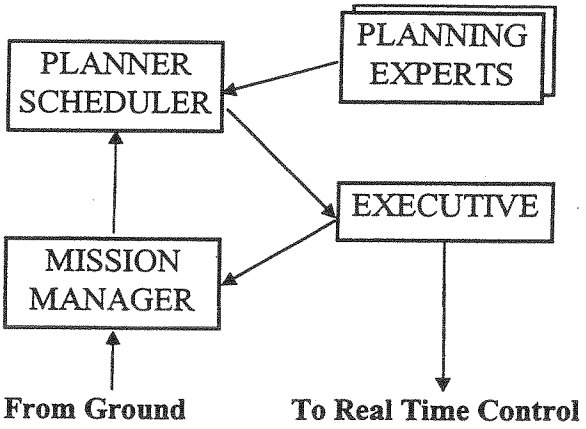


Figure 1: High Level Commanding

Execution flexibility: Procedures in a plan can be potentially executed in parallel. The plan explicitly represents and maintains temporal constraints between concurrent procedures. For example, a temporal constraint can express that procedure A must start from 30 to 60 minutes after procedure B, or that procedure B must execute while procedure C is executing, or that procedure A ends exactly when procedure C starts. PS ensures the consistency of the network of temporal constraints in the plan and infers *time ranges* during which a procedure can start and end. Unlike simple time tags, time ranges give EXEC the flexibility to compensate for execution delays caused by locally recoverable failures.

Robustness: an on-board planner can also make fault protection simpler and more robust than traditional sequencing. In the traditional approach, a sequence is infrequently uplinked to a spacecraft and therefore needs to include contingencies to handle a wide variety of failure conditions. In a "fail operational" scenario (e.g., a scenario in which the spacecraft autonomously recovers from a fault) the on-board sequence must be restarted, it must command the assessment of the new execution conditions, and react conditionally on the basis of this assessment. Because of the large number of possible failure conditions and the low level of the instructions in a sequence, the size of a robust sequence can be very large and the effort needed to build it very high. This is why "fail operational" scenarios are avoided as much as possible in traditional mission and are usually confined to critical mission phases (e.g., Cassini Saturn Orbit Insertion, encounter). In the RA approach, plans are valid only for the execution conditions known at the time PS was invoked and therefore the sequence of real-time commands issued is simpler and smaller. When execution conditions differ so much from the initial assumptions

that local failure recovery is insufficient, execution of the plan stops and PS is asked for a new plan that takes into account the new situation. Dealing with fault conditions on an as-needed basis simplifies the solution of the fault protection problem.

Generating Plans from Goals

Figure 1 describes how the DS1 PS implements high-level commanding within the RA architecture. A long-term plan containing goals for the entire mission, the *mission profile,* is stored and maintained on-board by the *Mission*

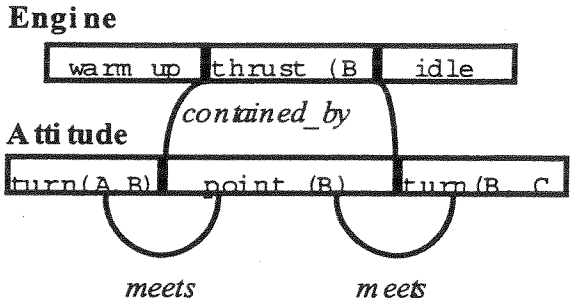


Figure 2: Example Timelines

Manager (MM). Ground operations interacts with MM to add, modify and delete goals in the mission. MM also responds to EXEC's requests for new plans by selecting a new set of goals from the mission profile, combining it with initial spacecraft state information provided by EXEC and sending it to PS. The time horizon covered by PS is typically two weeks during cruise and a few days during encounter. When a plan is ready, PS sends it to EXEC. When EXEC has almost completed execution of its current plan, it sends a new request to MM; this also happens when the EXEC is maintaining the spacecraft in standby mode after the occurrence of a major failure. In this case the initial spacecraft state will clearly identify which capabilities (if any) have been degraded by the fault. PS takes this information into consideration when generating the new plan.

Besides the mission profile, goals also come from other on-boards systems, the *planning experts*. For example, NAV communicates to PS of which beacon asteroids it needs pictures in order to estimate the current spacecraft position. The use of goals generated on board means that the spacecraft can modify its behavior, based on new information not known to ground. This capability is particularly important if the spacecraft has only infrequent contact with the ground or if ground reaction is too slow to take advantage of serendipitous opportunities. Goals generated on-board are incorporated in the plan during plan generation.

Plan representation

Both PS and MM use the same underlying system to represent, the *plan database.* A plan database is organized in several parallel *timelines*, each comprised of a sequence of *tokens.* A timeline describes the future evolution of a single component of the spacecraft's state vector. The set of tokens active at a given time represent the state vector value at that time. Goals and procedures are both represented as tokens. Each token consists of a state variable descriptor (specifying to which timeline the token belongs), a type (a symbolic representation of the goal or procedure and its parameters), a start-time, an end-time and a duration. Timelines can also represent renewable resources such as battery state of charge, non-renewable resources such as fuel, and aggregate resources (i.e., resources that can be allocated in parallel to several consumers) such as electric power. Temporal constraints synchronize resource allocation tokens with the corresponding consumer tokens. The type of the resource tokens indicate the amount requested and the modality of consumption (e.g., constant, linear depletion).

For example (Figure 2), a plan timeline may describe the state of the engine (warming up, thrusting, or idle) and another for the spacecraft attitude (e.g., pointing to a target, turning from target A to target B). The plan database also explicitly represents temporal constraints between tokens. These include constraints synchronizing tokens on separate timelines (e.g., "the spacecraft attitude must be pointing to target B while the engine is thrusting") and ordering tokens on the same or different time lines (e.g., "before the spacecraft can point to attitude A it must turn from its previous attitude B to A").

One important feature of the plan database is that decision variables (e.g., start or end time) and constraints among them are explicitly represented. The database then uses constraint propagation to infer valid ranges of values for variables and to detect inconsistencies (e.g., contradictory temporal constraints between tokens). This allows PS to concentrate on establishing constraints instead of selecting exact values for decision variables, an approach that often avoids over-commitment errors and therefore minimizes backtracking on earlier commitments.

The plan database can represent a plan at any stage of partial completion. Unlike complete plans, incomplete plans can have gaps between tokens on a timeline. Also, an incomplete plan may include an as-yet-unimplemented request for a constraint between tokens (see the section "The Domain Model"). The presence of these gaps and unfulfilled requests prompts PS to add tokens and constraints until the plan is complete. More representational details on the plan database can be found in [4].

The Domain Model

In a valid plan tokens must satisfy many constraints, including ordering (e.g., the catalyst-bed heaters must warm up for ninety minutes before using the reaction control thrusters), synchronization (e.g. the antenna must be pointed at the Earth during uplink,) safety (e.g. do not point the radiators within twenty degrees of the sun), and resource constraints (e.g. the MICAS camera requires fifteen watts of power). These are all expressed as temporal and parameter type constraint templates, or *compatibilities*, among token prototypes. The planning model is a set of compatibilities that must be satisfied in every complete plan. More formally, a.compatibility is a temporal relation that must hold between a *master token* and a *target token* whenever the master token appears in the plan. If the master token does not occur in the plan, the relation does not need to be satisfied. Compatibilities also specify relations between arguments in the master token type and value equivalence between arguments in the master and in the target token types. All compatibilities associated to a token are organized into a Boolean *compatibility tree.*

A simple compatibility tree is shown in Figure 3. It says that the state in which the MICAS camera is on must be preceded by a state in which it is turning on, and followed by one in which it is turning off. While the camera is on, it consumes fifteen watts of power.

```
(MICAS_On)
  :compatibilities
  (AND
    (met_by (MICAS_Turning_On))
    (meets  (MICAS_Turning_Off))
    (equal  (REQUEST (Power 15)))
```

Figure 3: A Compatibility Tree

Planning Algorithm

The planner searches in the space of incomplete or partial plans [5] with additional temporal reasoning mechanisms [6 and 4]. As with most causal planners, PS begins with an incomplete plan (given to it by MM) and attempts to expand it into a complete plan. The plan is complete when it satisfies all of the compatibilities in the plan model and there are no gaps on any timeline. The set of "defects" that need to be fixed in an incomplete plan in order for it to become complete is called the plan *conflicts.* Figure 4 summarizes the basic "conflict fixing" loop by PS. Each decision is typically made using heuristics and, when heuristic information is not particularly strong, using a uniform randomized rule (deterministic random number generator). If the wrong decision is made, PS will eventually reach a dead end, backtrack, and try a different path.

For example, consider one of the possible conflict types, *open compatibility.* An open compatibility is a temporal and parametric constraint that must exist between a *master* token already in the plan and a *target* token that may or may not be in the plan. For example, the compatibility `A meets B` is open if A is in the plan but B is not, or if both A and B are in the plan but the relation `A meets B` is not explicitly enforced. PS can satisfy an open compatibility with one of three resolution strategies. It can add the target token to the plan in such a way that it satisfies the temporal relation; it can adjust the start or end time of either the target or master token in order to satisfy the relation; or, it can decide that the relation will be satisfied by a token in the next planning horizon, and can therefore be ignored. These options are called *adding*, *connecting*, and *deferring*, respectively. Deferred compatibilities are maintained in the plan and carried forward to the next planning horizon as part of the initial state. PS will chose one of these options when it addresses an instance of an open compatibility conflict.

While plan has open compatibilities:
1.pick an open conflict;
2.select and apply a resolution strategy;
3.if no resolution is possible, backtrack

Figure 4: Planning Loop

Goal Prioritization

The overall mission goals depend on achieving a careful balance between potentially conflicting goals generated by independent sources (e.g., the science team, the navigation team). Conflicts typically arise because of over-subscription of limited resources (e.g., power, time). When a compromise is possible, PS appropriately distributes the use of available resources. When a compromise is not possible, then PS selects some of the lowest priority goals for postponement or outright rejection.

In PS tokens that have not yet been inserted onto a timeline, or *free token,* constitute a conflict category for which one of the possible resolution strategies is to reject the token. PS decides if the free goal token will be inserted. In DS1 PS does not explore all possible permutations of free token achievements but follows a statically assigned prioritization scheme (e.g., science goals have highest priority, followed by navigation goals and then by telemetry goals). This scheme, sufficient for DS1, avoids exponential backtracking but can yield sub-optimal solutions. Enhanced goal prioritization will be included in future PS versions.

Failure Response

RA provides a two level failure response — an immediate *reactive* response, and a longer term *deliberative*

response. This is typical of many autonomy architectures (e.g., Soar [7], Guardian [8]). The fast, real-time reactive behavior is implemented by EXEC and MIR. If this fails to solve the problem within the time and resource constraints of the current plan, then the failure can endanger future goals in the plan. In this case EXEC puts the spacecraft in standby, PS is called to assess the failure's impact on the remaining goals to decide how to best proceed. The deliberative response also addresses "advantageous failures" (e.g., serendipitous discoveries) and is the basis for enabling fundamentally new types of science missions.

This two level response results in simpler and more robust plans facilitating spacecraft commanding. The plans are simpler since they can address only the nominal case and trust that failures will be handled properly as they arise. Failures are either resolved by the reactive layer and allow the plan to continue, or cannot be resolved, in which case the plan breaks and the PS generates another nominal plan based on the new spacecraft state.

The plans are also more robust. This is partly due to the failure response mechanism, partly due to the hierarchical nature of the RA, and partly due to the plan representation. The hierarchy allows the tokens in the plan to describe fairly abstract procedures. The plan representation allows flexible start and end token times. Therefore EXEC has wide latitude in executing tokens, being allowed to respond to failures by retrying commands or trying alternate approaches. The extra failure response time needed is absorbed by the flexibility in the token's start and end times.

Conclusions

On-board planning is crucial for spacecraft autonomy. It can reduce mission costs and improve mission quality by allowing high-level commanding, enabling achievement of mission goals in the presence of failures without ground intervention, and taking advantage of fortuitous events. The DS1 mission marks the first on-board planner to fly on a spacecraft. The validation of this technology will open the way for future autonomous missions.

Acknowledgements

This work was performed in part at the Jet Propulsion Laboratory, California Institute of Technology, under contract to the National Aeronautics and Space Administration.

References

[1] Pell, B., Bernard, D., Chien, S., Gat, E., Muscettola, N., Nayak, P., Wagner, M. and Williams, B. 1996. A remote agent prototype for spacecraft autonomy. In Proceedings of the SPIE Conference on Optical Science, Engineering and Instrumentation.

[2] Pell, B., Gat. E., Kesing, R., Muscettola, N., and Smith, B., 1997., Plan Execution for Autonomous Spacecraft in *IJCAI* 97 (forthcoming).

[3] Williams, B.C., and Nayak, P.P., 1996. A model-based approach to reactive self-configuration systems. In *Proceedings of AAAI-96*, pp 971-978.

[4] Muscettola, N. 1994. HSTS: Integrating planning and scheduling. In Fox, M., and Zweben, M., eds, *Intelligent Scheduling*, Morgan Kaufman.

[5] Weld, D.S., 1994. An Introduction to Least Commitment Planning, *AI Magazine* Winter 1994.

[6] Allen, J.F. and Koomen, J.A. 1983. Planning using a temporal world model. *IJCAI 83*. pp. 741-747.

[7] Tambe, M., Johnson, W.L., Jones, R.M., Koss, F., Laird, J.E., Rosenbloon, P.S., and Schwamb, K. 1995. Intelligent agents for interactive simulation environments. *AI Magazine,* 16(1):15-39.

[8] Hayes-Roth, B. 1995. An architecture for adaptive intelligent systems. *Artificial Intelligence 72.*

CIRCA and the Cassini Saturn Orbit Insertion: Solving a Prepositioning Problem

David J. Musliner and Robert P. Goldman
Automated Reasoning Group
Honeywell Technology Center
3660 Technology Drive
Minneapolis, MN 55418
{musliner,goldman}@htc.honeywell.com

Introduction

The Cooperative Intelligent Real-time Control Architecture (CIRCA) was designed for mission-critical applications requiring autonomous planning and control. As space technology becomes mature and budgets shrink, NASA is looking towards precisely this type of autonomy to support future missions including the New Millennium Deep Space One probe and the Mars Rover projects. We are actively investigating the application of CIRCA to a variety of domains including spacecraft planning and control and autonomous aircraft control. In this paper, we discuss one particularly challenging type of planning problem that arises in mission-critical applications, drawing on an example from the Cassini Saturn mission.

Prepositioning problems arise when certain actions must be taken to preposition assets or otherwise prepare for contingencies, before those contingencies could possibly occur. In the Cassini example, a backup inertial reference unit (IRU) must be preheated long before an engine burn is planned, so that if the primary IRU fails during the burn, the backup will be immediately available. The IRU preheating operation is a "prepositioning action." To build plans that include this type of prepositioning, a planner must "look ahead," recognize the contingency, and identify appropriate prepositioning actions. CIRCA's new Dynamic Abstraction Planner (DAP) efficiently builds plans that include prepositioning.

This paper is not meant to be an introduction to CIRCA; instead, our goal is to describe how CIRCA can address the Cassini prepositioning problem, and discuss the various strengths and weaknesses of the approach. Accordingly, we refer readers to other publications (Musliner, Durfee, & Shin 1993; 1995; Goldman *et al.* 1997) for CIRCA overviews and details on the planning algorithms. This paper begins by describing key elements of prepositioning problems in general, and then gives the details of the simplified Cassini problem. We then present the CIRCA solution to this problem. We review how other systems attempt to solve this type of problem, comparing their features with CIRCA, and conclude with a discussion of the more general directions for future work on CIRCA.

Prepositioning Problems

Prepositioning problems arise when actions must be taken to prepare for contingencies, *before* those contingencies could possibly occur. For example, military prepositioning involves the movement of assets to various forward deployment areas when no conflicts are in progress, in anticipation of future conflicts in the area. In the Cassini example, a backup inertial reference unit (IRU) must be preheated long before an engine burn is planned, so that if the primary IRU fails during the burn, the backup will be immediately available. These examples share several common features that we use to characterize prepositioning problems:

Exogenous Events — Without events beyond the system's control there would be no need for prepositioning. If the domain is fully controllable, prepositioning becomes simply establishing preconditions for planned actions. Prepositioning is different because it involves anticipating contingencies outside of the system's control.

Potential Failure — It must be possible for the system to fail, or reach an undesirable state/outcome, if the prepositioning action is not performed correctly. The plant must blow up, the spacecraft miss its orbit insertion, etc., else there is little motivation to preposition.

Temporal Information — By definition, prepositioning actions must be performed in a timely fashion, so that they are completed before the contingency can occur.

These problem characteristics place stringent requirements on planners that must address prepositioning

ing problems: a suitable planner must be able to represent and reason about exogenous events, nondeterminism, failure, and time. Because prepositioning actions must be taken before the disturbance occurs, reactive approaches are not sufficient; projection is needed. On the other hand, because *external* contingencies are involved, "classical" AI projective planners are not sufficient; they assume the planning agent is the only active part of the domain.

CIRCA strikes a particular balance along the spectrum of representational power vs. computational complexity. This balance point was explicitly chosen to represent the information necessary for mission-critical planning problems without unnecessary complexity.

CIRCA and Prepositioning Problems

In particular, CIRCA provides the following features that can be applied to prepositioning problems:

Event and Temporal Transitions — CIRCA can model two types of exogenous changes, event transitions and temporal transitions. Both are modeled as completely outside of the agent's control. Events are instantaneous changes, while temporal transitions represent processes with temporal extent.

Explicit Failure — CIRCA models a distinguished failure state which it must plan to avoid at all costs. CIRCA builds plans that guarantee that failure cannot be reached. This is the key feature that makes CIRCA suited to mission-critical applications.

Simplified Temporal Model — The temporal model in CIRCA is reduced to the minimum amount of information necessary to build plans that ensure safety. Transitions are only assigned worst-case timing values. Actions that are under the planner's control are assigned maximum delays; the system can rely on these actions taking effect by the maximum delay. On the other hand, events and temporals have minimum delays; the system can rely on these transitions *not* occurring before the delay elapses. Together, these upper and lower bounds impose the minimum and maximum limits on the dwell time in a state. For example, a planned action can be assigned a maximum delay value D to ensure that the system can dwell in a state for no more than D time units before the action must occur. The CIRCA execution engine (the Real-Time Subsystem) will enforce this execution timing requirement.

State-space Projection — CIRCA's state-space planner builds plans by interleaving a forward simulation of the environment with the selection of actions for each simulated state. Lookahead search heuristics allow the planner to make intelligent action choices, and smart backjumping improves the search for a useful plan when lookahead is not sufficiently prescient.

The Cassini Prepositioning Problem

The Cassini prepositioning problem was originally outlined by Erann Gat in his paper "News From the Trenches: An Overview of Unmanned Spacecraft for AI" (Gat 1996). The problem concerns the Saturn orbit insertion maneuver planned for the Cassini spacecraft. The spacecraft has a narrow time window during which it must fire its thrusters to decrease its speed and enter the desired Saturn orbit. To successfully navigate during a thruster burn, the Cassini spacecraft must have an inertial reference unit (IRU) warmed up and functioning. The spacecraft has a primary and a secondary IRU. The problem is to foresee the possibility of a primary IRU failure during the orbit insertion burn and warm up both IRUs early enough that they will be available for navigation during the burn. If the primary fails but the backup has been warmed up, the backup can be switched in and the burn can continue uninterrupted.

Gat originally claimed that "there is currently no AI planner that can figure out, given this information, that it is a good idea to turn on the backup IRU before orbit insertion begins so that the burn doesn't have to be terminated if the primary IRU fails." We claim, on the contrary, that CIRCA can easily build this plan, given quite sparse information about the problem. In fact, it is precisely the sort of planning that CIRCA is good at. In the following section, we give full details on one version of the domain encoding that CIRCA can solve, discussing several necessary quirks of the representation as well as its strengths.

CIRCA Solving the Cassini Problem

Figure 1 shows the domain encoding we used to solve the Cassini prepositioning problem. Of course, this is a highly simplified representation of the real problem, focused only on illustrating how CIRCA decides to preheat the backup IRU (IRU2). In that sense, this domain encoding is like a blocks-world problem; we are not claiming it captures the real complexity of the Cassini domain.

The representation is fairly straightforward, specifying the initial state and the possible events, temporal transitions, and control actions. For CIRCA, the goal of avoiding failure is always implicit. The `*goals*` expression describes "task-level" goals that the planner should try to achieve, but which are not mission-critical (there are none in this domain). We have made the `(engine on)` goal a safety-critical "control-level" goal using the `fail_if_dont_burn` transition. This transition specifies that, if the engine stays off for

```
(setf *goals* nil)
(make-instance 'action :name "warm_IRU1"
      :preconds '((IRU1 off))
      :postconds '((IRU1 on))
      :delay 60)
(make-instance 'action :name "warm_IRU2"
      :preconds '((IRU2 off))
      :postconds '((IRU2 on))
      :delay 60)
(make-instance 'action :name "engine_on"
      :preconds '((engine off))
      :postconds '((engine on))
      :delay 1)
(make-instance 'action :name "select_IRU1"
      :preconds '((IRU1 on))
      :postconds '((active_IRU IRU1))
      :delay 1)
(make-instance 'action :name "select_IRU2"
      :preconds '((IRU2 on) (IRU1 broken))
      :postconds '((active_IRU IRU2))
      :delay 1)
(make-instance 'temporal :name "fail_if_burn_with_broken_IRU1"
      :preconds '((engine on)(active_IRU IRU1)(IRU1 broken))
      :postconds '((failure T))
      :delay 5)
(make-instance 'temporal :name "fail_if_burn_without_guidance"
      :preconds '((engine on)(active_IRU none))
      :postconds '((failure T))
      :delay 1)
(make-instance 'temporal :name "fail_if_dont_burn"
      :preconds '((engine off))
      :postconds '((failure T))
      :delay 1000)
(make-instance 'event :name "IRU1_fails"
      :preconds '((IRU1 on))
      :postconds '((IRU1 broken)))
(setf *initial-states* (list
      (make-instance 'state
            :features '((failure nil) (engine off) (IRU1 off) (IRU2 off)
                  (active_IRU none)))))
```

Figure 1: Simple Cassini prepositioning domain description.

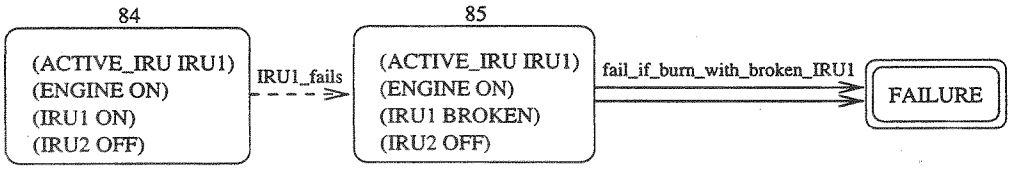


Figure 2: Partial state space during planning, where failure is reachable.

1000 time units, the system may fail catastrophically. Since CIRCA builds plans that must guarantee to avoid (`failure T`), the final plan will definitely turn the engine on.

To illustrate how CIRCA's planner solves this problem, we'll trace a small portion of the search activity. Consider the partial plan shown in Figure 2. In this sequence, the planner considers the `IRU1_fails` event transition leading out of state 84 to state 85. From this state, a temporal transition (`fail_if_burn_with_broken_IRU1`) leads quickly to failure, which the planner must prevent. The planner recognizes that it cannot take the `warm_IRU2` and `select_IRU2` actions in time to beat (or "preempt") the potential failure. In other words, the planner has projected a failure and its temporal model has shown that no corrective actions are available once state 85 is reached. The only possible solution is to take an earlier, prepositioning action aimed solely at preventing state 85 from ever occurring. The planner labels state 85 as a (transitive) failure state for future reference, and backtracks. The smart backjumping function is able to recognize the first decision that made state 85 reachable, backtrack to that point, and alter that and future decisions to preheat IRU2, avoiding state 85.

Note that this plan fragment is actually a hypothetical example — the real DAP planner running on the domain in Figure 1 generates a plan in somewhat more complex ways because it interleaves the process of splitting (refining) the abstract state descriptions with the process of choosing actions for states. In fact, DAP recognizes the potential failure and plans the prepositioning action at a much earlier point in the search. Space limitations do not permit us to show the actual DAP planning sequence, but the final plan is shown in Figure 3. There are several intriguing aspects of this plan and the domain encoding that deserve brief discussion.

Representation Hacks and Plan Quirks

IRU2 Cannot Fail Observant readers will have noted that IRU2 cannot fail according to Figure 1. We have omitted this transition because if IRU2 can fail, there is no guaranteed safe plan because both IRUs can fail, causing the burn to possibly fail. Since CIRCA will only produce safe plans, adding an `IRU2_fails` event leads the planner to (correctly) conclude that the domain is overconstrained and unsolvable, and that some performance tradeoffs must be made. One typical performance tradeoff, for example, would be to ignore certain lower-probability transitions such as the failure of a backup system. The current DAP state-space planner is not yet integrated with a higher-level tradeoff engine.

Preferring IRU1 Another unusual aspect of the domain encoding is the inclusion of (`IRU1 broken`) in the preconditions of the `select_IRU2` action. This has the effect of making the planner "prefer" to select IRU1, since it cannot use IRU2 unless IRU1 is broken. If this condition is omitted, the planner is smart enough to exploit the fact that IRU2 never fails, by always selecting IRU2. This has the beneficial effect of making IRU1 irrelevant, but it also sidesteps the prepositioning problem. Thus the extra precondition in the `select_IRU2` action is an idiomatic way of forcing the planner to respect a primary/backup ordering over the IRUs.

State 127 As is often the case with AI programs, DAP can arrive at non-intuitive but correct plans. In state 127 of Figure 3, IRU1 is broken but selected and the engine is off. Surprisingly, the planner chooses to turn on the engine and *then* select IRU2, rather than vice versa. According to the domain model of Figure 1, the chosen order is acceptable because the `select_IRU2` action can be guaranteed to complete before the temporal transition to failure (`fail_if_burn_with_broken_IRU1` from state 39) can possibly occur.

Related Work

Other researchers have attempted to extend "classical" AI planning techniques to handle problems like the prepositioning problem. One approach to prepositioning problems is offered by "conditional planners." These are conventional AI planning systems that have been extended so that they can generate plans with embedded "if-then-else" structures. The first such planner was Warren's Warplan-C (Warren 1976). More recently, causal link planners have been extended to conditional planners by Peot and Smith (1992), Pryor and Collins (1996), and Goldman and Boddy (1994). The UWL SENSp softbot planner also did some conditional planning (Etzioni *et al.* 1992).

These conditional planners all make the same extension to classical AI planners: they permit nondeterministic actions that have multiple possible outcomes. For example, the Plinth image processing planner (Goldman & Boddy 1994) has an operator for running a neural net region classifier that may succeed or fail. Conditional planners attempt to find plans that will achieve the goal for all possible outcomes of conditional actions. When Plinth constructs a plan using the neural net region classifier, it considers the possibility that the classifier may fail and adds a contingency plan to handle that eventuality.

Conditional planners do not provide a fully satisfactory solution to the prepositioning problem because

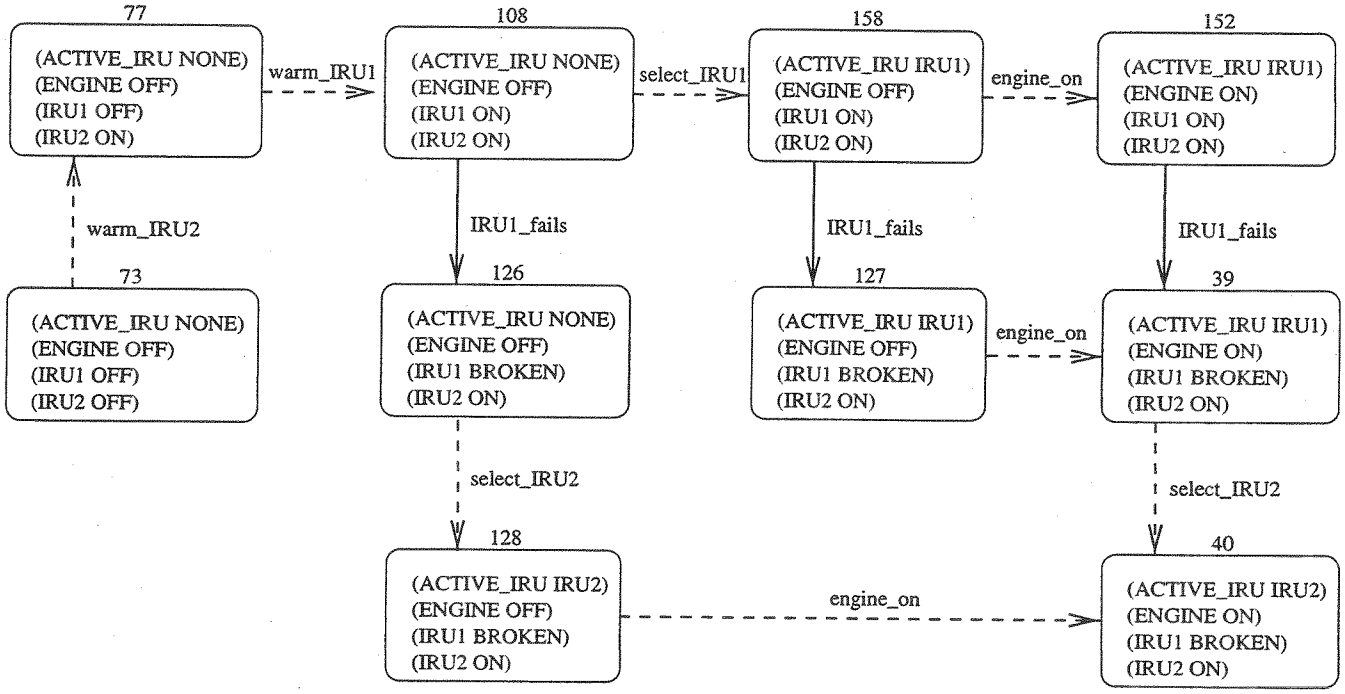


Figure 3: The final Cassini state space.

they do not provide a natural model of the true problem. The true problem is to preposition assets in order to respond to foreseeable disturbances. However, conditional planners keep the classical planning assumption that the planning agent is the only actor. Because of this assumption, exogenous events can only be encoded as an outcome of one (or many) of the planning agent's actions. This encoding complicates the domain knowledge engineering and yields cumbersome encodings in cases where events can occur at multiple times. Furthermore, these planners do not handle the temporal aspects of the prepositioning problem.

Jim Blythe has developed a planner, Weaver, that more directly addresses the problem of reacting to exogenous events (Blythe 1996). This planner generates a plan for a main line of execution, then incrementally adds significant domain events and plans reactions to these events. However, Weaver, like the conditional planners, does not address temporal aspects of the prepositioning problem.

There are probabilistic and decision-theoretic planners that provide facilities similar to those we have discussed above (Drummond & Bresina 1990; Kushmerick, Hanks, & Weld 1993; Draper, Hanks, & Weld 1993; Dearden & Boutilier 1997). We agree that such approaches are, broadly speaking, the Right Thing. However, as a practical matter, the need for a probabilistic model adds a burden over and above the modeling needs of non-stochastic approaches, and does not (in our experience at least) appeal to domain users.

We are examining ways to extend CIRCA to handle probabilistic measures of uncertainty, but in a minimalist way. Atkins *et al.* (1996) have also worked on developing a probabilistic CIRCA planner. Their planner is intended to reason about the likelihood of various transitions, so that the planner can use its limited resources to consider the most likely transitions first. This project is complicated by the fact that the CIRCA model is not Markovian: it matters what the preceding state was, and how long one remained there. As far as we know, none of the other probabilistic approaches address the temporal aspects of the problem.

Kabanza *et al.* (1997) have developed a planning method for reactive agents that is similar to the original CIRCA. Their architecture differs in emphasis, however. The NFAs it constructs are "clocked:" they make transitions at times that are the least common denominator of all possible transitions. This scheme will suffer a state space explosion in domains where there is a wide range of possible transition delays, like those to which CIRCA has been applied. Kabanza's group has concentrated on developing a more flexible notation for goals than those used by CIRCA, but they do not make the same distinction between safety and goal achievement.

Conclusions

Our current work on CIRCA extends in four directions: (1) planning with more expressive domain models; (2) determining how best to incorporate CIRCA

into an autonomous agent; (3) distinguishing significant (and insignificant) disturbances and (4) developing a more efficient state-space planner (see Goldman *et al.* (1997)).

Currently, CIRCA's planner only chooses either to preempt exogenous processes or to allow them to happen; it cannot *rely* on them happening, because it does not have an upper bound on their delay.This expressive limitation makes it difficult to represent intentional processes like the warming up of an IRU. Currently, we must do one of two things: (1) make "warming up the IRU" an atomic action that takes a very long time or (2) encode the fact that the IRU is warming into the feature space and have a temporal transition from warming to warm. The first alternative is undesirable because it makes it impossible for the CIRCA agent to do anything else while it is waiting for the IRU to warm. The second option is actually worse, because it makes it impossible for CIRCA to rely on the IRU ever becoming warm. To make the second option work correctly, we are extending the CIRCA model to include "reliable temporals" that have upper bounds on their times of occurrence.

We noted above that our Cassini domain encoding incorporates the design decision that it is not worth worrying about two consecutive IRU failures. Ideally, one could use a probabilistic world model to automatically identify significant eventualities, rather than having to identify them manually (cf. Atkins *et al.* (1996)). However, we want to do this without the domain engineering overhead of constructing a full Markov process model; constructing this model is at least as hard as identifying the eventualities for the planner. We are currently examining a technique for model pruning that assumes independent component failures (an assumption commonly used in reliability engineering) and calculating only order-of-magnitude likelihoods on CIRCA state space graphs.

Another limitation is CIRCA's lack of a system clock. Currently, CIRCA can only reason about duration relative to the time it enters a particular state. To meet deadlines that are related to a global clock (e.g., the actual Cassini orbital burn time), CIRCA's RTS executive must be able to act at an appropriate time relative to a planned future event. We do not want to abandon the unclocked executive, because inclusion of global time into the state space can cause it to explode. We are exploring how to coordinate the CIRCA state-space planner with an overall mission planner/scheduler. The mission planner/scheduler will be able to provide signals indicating important global times to the CIRCA RTS. The RTS will then detect these signals like any other state feature. Our preliminary investigations suggest that we can detect the need for such features through search failures in the AIS.

The higher-level CIRCA mission planner/scheduler or "task-level planner" (Musliner 1993) plays a major role in managing the state-space planner's activities and the sequence of plans executed by the RTS. The task-level planner determines which lower-level planning jobs the DAP planner should work on, what safety-critical and best-effort goals should be pursued, what performance tradeoffs should be made, what low-probability transitions should be ignored, etc. This higher-level planner module is an area of active research, with great potential to complement CIRCA's DAP state-space planner and RTS, yielding a powerful, domain-independent intelligent real-time control system.

Acknowledgments This work was supported by the Defense Advanced Research Projects Agency under contract DAAK60-94-C-0040-P0006.

References

Atkins, E.; Durfee, E. H.; and Shin, K. G. 1996. Plan development using local probabilistic models. In *Proc. Conf. on Uncertainty in Artificial Intelligence*, 49–56.

Blythe, J. 1996. A representation for efficient planning in dynamic domains with external events. in the AAAI workshop on "Theories of Action, Planning and Control: Bridging the gap".

Dearden, R., and Boutilier, C. 1997. Abstraction and approximate decision-theoretic planning. *Artificial Intelligence* 89(1–2):219–283.

Draper, D.; Hanks, S.; and Weld, D. 1993. Probabilistic planning with information gathering and contingent execution. Technical Report 93-12-04, Department of Computer Science and Engineering, University of Washington, Seattle, WA.

Drummond, M., and Bresina, J. 1990. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proc. Nat'l Conf. on AI*, 138–144.

Etzioni, O.; Hanks, S.; Weld, D. S.; Draper, D.; Lesh, N.; and Williamson, M. 1992. An approach to planning with incomplete information. In *Proc. Int'l Conf. on Principles of Knowledge Representation and Reasoning*, 115–125.

Gat, E. 1996. News from the trenches: An overview of unmanned spacecraft for AI. In Nourbakhsh, I., ed., *AAAI Technical Report SSS-96-04: Planning with Incomplete Information for Robot Problems.* American Association for Artificial Intelligence. Available at `http://www-aig.jpl.nasa.gov/home/gat/gp.html`.

Goldman, R. P., and Boddy, M. S. 1994. Conditional linear planning. In *Proc. Int'l Conf. on AI Planning Systems.*

Goldman, R. P.; Musliner, D. J.; Krebsbach, K. D.; and Boddy, M. S. 1997. Dynamic abstraction planning. In *Proc. Nat'l Conf. on AI*, 680–686.

Kabanza, F.; Barbeau, M.; and St-Denis, R. 1997. Planning control rules for reactive agents. Technical Report 197, Computer Science Dept., University of Sherbrooke.

Kushmerick, N.; Hanks, S.; and Weld, D. 1993. An algorithm for probabilistic planning. Technical Report 93-06-03, Department of Computer Science and Engineering, University of Washington, Seattle, WA. to appear in *Artificial Intelligence*.

Musliner, D. J.; Durfee, E. H.; and Shin, K. G. 1993. CIRCA: a cooperative intelligent real-time control architecture. *IEEE Transactions on Systems, Man and Cybernetics* 23(6):1561–1574.

Musliner, D. J.; Durfee, E. H.; and Shin, K. G. 1995. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence* 74(1):83–127.

Musliner, D. J. 1993. *CIRCA: The Cooperative Intelligent Real-Time Control Architecture.* Ph.D. Dissertation, University of Michigan, Ann Arbor. Available as University of Maryland Computer Science Technical Report CS-TR-3157.

Peot, M. A., and Smith, D. E. 1992. Conditional nonlinear planning. In *Proc. Int'l Conf on AI Planning Systems*, 189–197.

Pryor, L., and Collins, G. 1996. Planning for contingencies: A decision-based approach. *Journal of Artificial Intelligence Research* 4:287–339.

Warren, D. H. 1976. Generating conditional plans and programs. In *Proceedings of the AISB Summer Conference*, 344–354.

Application of Artificial Intelligence to Planning and Scheduling for Operations On-board the Russian Mir Space Station

Barrios/Jill C. Novak, USA/Felesha Robertson
Flight Planning & Tool Development Group
NASA Lyndon B. Johnson Space Center
Houston, TX 77058
jill.c.novak1@jsc.nasa.gov
felesha.l.robertson1@jsc.nasa.gov

Abstract

In preparation for International Space Station (ISS) planning, planners at NASA-Johnson Space Center (JSC), NASA-Marshall Space Flight Center (MSFC), and Russian Space Agency (RSA) conducted a demonstration to test the planning process and scheduling tools which are expected to be used. The scheduling tools tested included the Consolidated Planning System (CPS) and Intelligent Flight Activities Planner (IFAP). IFAP is designed to use artificial intelligence in conjunction with the planning and scheduling capabilities of CPS. As a result, the demonstration provided experience with scheduling on-board the Russian Mir space station which will be used as the foundation for future mission planning.

Phase 1 Overview

Phase 1 is a NASA program encompassing 11 space shuttle flights over a four-year period. It uses existing assets - primarily U.S. shuttle Orbiters and the Russian Space Station Mir - to build joint space experience and start joint scientific research.

International Space Station Phase 1 serves as a four-year prologue to station assembly in Phases 2 and 3. NASA and its international partners are finding that Phase 1 is a valuable precursor to learning about all aspects of living and working in low-Earth orbit.

NASA and Russian engineers, designers, technicians, and flight crews work together to achieve a common goal by making many small, practical decisions on a daily basis, melding their different work styles into a unified plan.

Shuttle-Mir is a complicated interlocking program incorporating the very different working styles and philosophies of the U.S. and Russian space agencies and their international partners. The job is made manageable by the cooperation of mission planners, hardware engineers, flight crews and others who are united in their dedication to the program's success.

Phase 1 Demonstration

The purpose of the Phase 1 Planning Demonstration was to produce an integrated systems and payload timeline for a two-week period during a Mir long duration mission using the planning process and tools developed for the International Space Station (ISS) program. The demonstration aided in the development of planning tools and methods of interfacing required to facilitate the transition from Phase 1 to Phase 2 of the ISS program. The demonstration also decreased the risks of encountering planning difficulties in Phases 2 and 3. It was planned that the demonstration would be conducted during a time period when no major events (i.e. other earth-to-orbit vehicles (ETOV) present, extra-vehicular activities (EVA's), and other activities) were occurring.

In the process of building two one-week plans for the Mir, planners at all centers would not only verify the planning processes, but they would also establish new ones in an effort to increase the efficiency of plan development. Further, by testing the capabilities of the tools, planners at all centers would be trained on how to use the software and would find software-related problems that would hinder or prevent planning product development.

Preparation

The goal of the Phase 1 Demo Team was to develop two 1-week plans (21 July - 3 August 1997) for ongoing Mir operations using process concepts that have been developed for ISS. It would be accomplished by using the work of planners at NASA-JSC, NASA-MSFC, and RSA. RSA planners would be responsible for the Russian systems and payloads plan; NASA-MSFC planners would be responsible for the US payloads plan and for overall payload integration; NASA-JSC would be responsible for overall plan integration. Two JSC consultants were in Moscow at all times during the Demo for software and process support.

Extensive work on procedures, database standards, and formats was done prior to the start of the Demo; participants from all planning centers were involved in the development of those products. A "dry run simulation" was conducted in May of 1997 as a preliminary testing of the tools and processes, and the actual Demo began on 24 June 1997. All planners used the Consolidated Planning System (CPS) to develop Demonstration timelines, although high level plans were also built using the

Scheduling and Planning Optimization Model (SPOM) (RSA) and IFAP (NASA-JSC).

Extensive preparation was coordinated before the Demonstration on several levels. The overall Demo team prepared itself through weekly International Execute Planning Team telecons. Individual centers prepared themselves as well with varied degrees of training. No amount of preparation fully readied the team for the contingencies that occurred on the Mir space station during the Demonstration's timeframe.

Execution

On 25 June 1997 a Russian Progress supply ship collided with the Mir space stations during a manual redocking exercise. The collision caused damage to the solar arrays and the Spektr module began to lose pressure. The hatch between Spektr and the rest of the station was sealed, but in order to do so, many cables had to be cut including cables that supplied power to the rest of the station.

The IEPT made a decision to reflect the real plan for Mir operations. This required planners to deviate from the nominal short term planning (STP) process to produce contingency STP's on schedule. A significant amount of time and effort was spent trying to reflect the Mir contingency plans.

Phase 1 Demonstration Products

The On-orbit Operations Summary (OOS) was the product that provided a planning overview of the demonstration and was the starting point from which the short-term scheduling process began. It showed daily listings of major and significant activities and additional information that was used to gauge whether activity requirements were being met. (The level of definition for the ISS OOS is still under discussion).

The Short Term Plan (STP) was the major demonstration planning product. It served several purposes in displaying activity information. It was the product used for review and approval of all system and payload activities. It was broadcast to all demonstration participants to inform them of what events were scheduled to be performed during the demonstration. The STP contained all data needed for execution of MIR activities and was used by ground controllers. The STP is also the plan from which executable products were built and uplinked to MIR for execution.

The STP's covered 7 days of operations for the demonstration. (This is still under discussion for ISS operations). They included all on-board activities and those ground activities required to directly support on-board activities. A unique feature of the STP is the fact it includes ground and automated activities as well as crew activities.

About the Artificial Intelligence

The Consolidated Planning System (CPS) is the planning tool that provides the planner with functions such as activity scheduling, resource utilization scheduling, timeline development and analysis, core/payload timeline integration/separation, condition constraints scheduling, and scheduling conflict detection. These CPS functions are very useful to the Operations Planners. The ISS will have limited, non-constant resources; therefore, it is important that the planners be able to define activity resource requirements and be able to track resource availability/utilization information so as not to exceed resource availability. In addition, activity condition needs such as communication coverage and lighting pose constraints to on-orbit activities. CPS allows the planner to define the activity condition constraints for automatic scheduling against the conditions as well as the resource and temporal constraints. Also available is a conflict detection capability of CPS that provides the planner with information as to why an activity failed to schedule. The planner should then analyze the CPS-generated conflict detection data to determine how to get the activity to schedule on the timeline. These are just a few reasons why the CPS tool and its capabilities are important and useful to the planning community.

In addition, there is a software application that provides flexible, automatic scheduling and planning engines that works in cooperation with CPS called the Intelligent Flight Activities Planner (IFAP).

IFAP uses artificial intelligence techniques to capture the expertise of planners and to provide comprehensive, interactive scheduling and planning assistance. IFAP can automatically plan taking into consideration resources, conditions, bands, resource groups, and temporal constraints. IFAP allows the planner to customize the scheduling engine by selecting from a variety of scheduling methods. The IFAP planning engine allows the user to plan activities and sequences to the day rather than permanently scheduling them to the minute and second. The user can convert planned data into scheduled data using the Planning Window provided by IFAP.

Dynamic explanation capabilities give the planners insight into the reasoning that produced the schedule, including the reasons for particular resource assignments, why start dates and end dates were changed, and the scheduling order used.

The Intelligent Flight Activity Planner serves as a planning tool with a user friendly interface, that meets the needs of the Shuttle and Space Station Programs. The ISS planners have committed to use IFAP for the Space Station command execution planning. IFAP is expected to increase the productivity of the ground planning team.

One of the strongest uses of artificial intelligence is through the ability the planner has of setting up and enforcing scheduling rules in IFAP. Rules are used during scheduling to enforce certain restrictions on the placement of the activities. A rule works by evaluating a function for each proposed location of the activity being placed. The value the function returns dictates how the activity is scheduled.

CPS/IFAP resides on IPS (Integrated Planning System). IPS is a UNIX-based, client-server, distributed system that is being developed to support several planning and analysis functions. IPS currently exists as a network of individual IBM AIX Power PC workstations in a file server architecture with a unified set of software tools.

Together, CPS/IFAP is designed to provide an effective combination of computational analysis power with man-in-the-loop decision making. The planner relies on the software to manage resource use and external constraints while focusing his/her attention on political and other decisions which require human interaction and flight experience.

This document will define the application problems encountered during the Phase 1 Demonstration. In addition, benefits of the planning and scheduling application, as well as requirements we would like to incorporate with CPS/IFAP will be addressed.

Process Problems and Lessons Learned

The following observations are process problems that were documented as lessons learned during the Phase 1 Demonstration. In addition, recommend solutions, made by NASA-JSC, NASA-MSFC, and RSA Planners, to correct the problems before executed use onboard the ISS are provided:

Tools/Platform. Delays in product delivery were due to problems experienced with some of the hardware not having memory large enough to handle the amount of data used in the demonstration. Also, the Oracle database was not configured to handle the amount of data used in the demonstration. Planners have recommended to assess the capacity, performance, & configuration requirements on the Integrated Planning System (IPS) for ISS.

As the activity database grew in size, major decreases in software performance were experienced. In some cases, the software was rendered useless. Planners determined that the platforms intended for ISS operations will require increases in memory and processor speed necessary to handle the volume of planning data expected.

The lack of several key capabilities in CPS (planning, conditions scheduling, annotations deletion, etc.) required significant user workarounds and development of additional user programs (SQL scripts). This added to the heavy workload. A minimum complement of tool capabilities (planning and scheduling) is absolutely essential for long term ISS operation.

Also, CPS lacks several of the mandatory requirements for distributed planning and many of the features necessary to adequately represent user requirements. There are plans to continue to submit change requests to CPS and strive to have all mandatory requirements implemented. Using scheduling software for long term planning should be avoided.

IPS-REM. Software development and Demonstration planning were both conducted on the IPS-REM (IPS-Remote Extension Moscow) computer without any coordination. Early in the Demo, some development work lead to the complete loss of the Oracle database on the machine. A recommended solution to this problem would be to develop a mutually agreed upon configuration management plan for the IPS-REM.

Database development. RSA planners and NASA-JSC planners developed parallel planning databases in both SPOM and CPS in order to produce the planning products for the Demonstration since no interface between the programs was available. Prior to the execution of ISS, RSA and JSC planners should jointly develop SPOM requirements for integration with CPS.

Database Standards. A significant amount of time was spent in developing mutually agreed to database standards.
Planners agreed that the development and adherence to database standards was critical to effective multi-center planning and consistent report generation. The time spent on these standards was necessary for team coordination. This effort should be repeated for future endeavors.

Translations. A large amount of time was spent waiting for the translation of critical documents. "In-house" translations by members of the planning team proved effective for understanding general concepts, but were insufficient for the understanding of specific constraints. Planners have recommended to minimize translation requirements during STP development and real-time execution by comprehensive database development with standards, glossaries of terminology, and effective interfaces between CPS and SPOM databases.

Development community. The development community was not involved in the Demo. As a result, users were not up to date on the capabilities of the platform or the software available. Inclusion of developers in the Moscow consultant team proved invaluable in recovery from server performance and database problems. It was recommended that future projects should include the development community from start through execution stage.

File management development. A significant effort was expended in developing file transfer and archival processes that proved useful in recovering from various platform and database problems. Planners have decided to continue developing mutually agreed upon processes for file management. Any effort that the development community can make towards more effective file transfer and archival could facilitate operations in the future.

A significant amount of personnel effort was expended in managing file transfers, maintaining the "loop" of data flow traffic, and keeping up-to-date archives during the STP development process. Platform enhancements are desired in order to make the file management process more efficient. The planning team composition should account for the importance and effort involved in data management.

Plan generation software. Each group used a different planning program for OOS generation. Lack of an electronic interface between any of these programs forces JSC planners to manually enter activities into IFAP. This lead to the inability to form an integrated OOS for the Demo. Planners determined that common software and a common database are essential to integrated planning. A robust, high-level planning software planning package is necessary for effective planning, and should be used by each center.

Planning Procedures. Several months were spent in developing, documenting, and agreeing on the procedures to be followed for STP development during the demonstration. These procedures were exercised and verified during a dry-run conducted a month before the actual demonstration. The procedures reflected the needs and views of the planning team as a whole, and the dry-run proved their effectiveness in a nominal situation. All future endeavors should follow this model in developing a joint planning process.

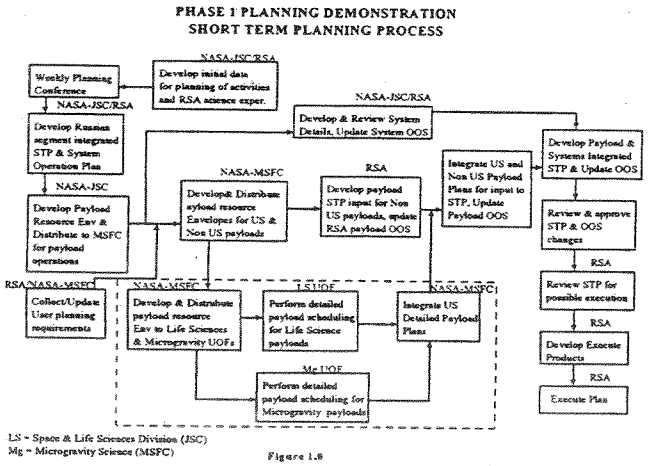


Figure 1.0

STP Process. STP process was serial in nature, and focused on detailed scheduling. See Figure 1.0. The anomalous environment of the planning demonstration required several departures from the process in order to produce effective plans on schedule. Planners recommended the following solutions:

1. The planning process must be flexible, allowing for iteration between systems and science planning communities.
2. Time zone differences between partner sites should be leveraged in order to provide more time in process.
3. Detailed scheduling should be deferred until as late as possible. A "just-in-time" scheduling approach should be utilized.
4. Review and re-examine the concept for resource envelopes. Any changes to the planning concept need to be accommodated by the resource envelope scheme.
5. Examine the "job jar" philosophy in its relation with the resource envelope concept. The current concepts appear to be incompatible with one another.
6. Re-examine the STP process regarding level of detail and responsiveness to real-time changes and management directives.

The Final Products

An example of the STP from weeks 1 and 2 can be found at the end of this report. These examples represent the first page of each weekly STP from the Phase 1 Demonstration.

Conclusion

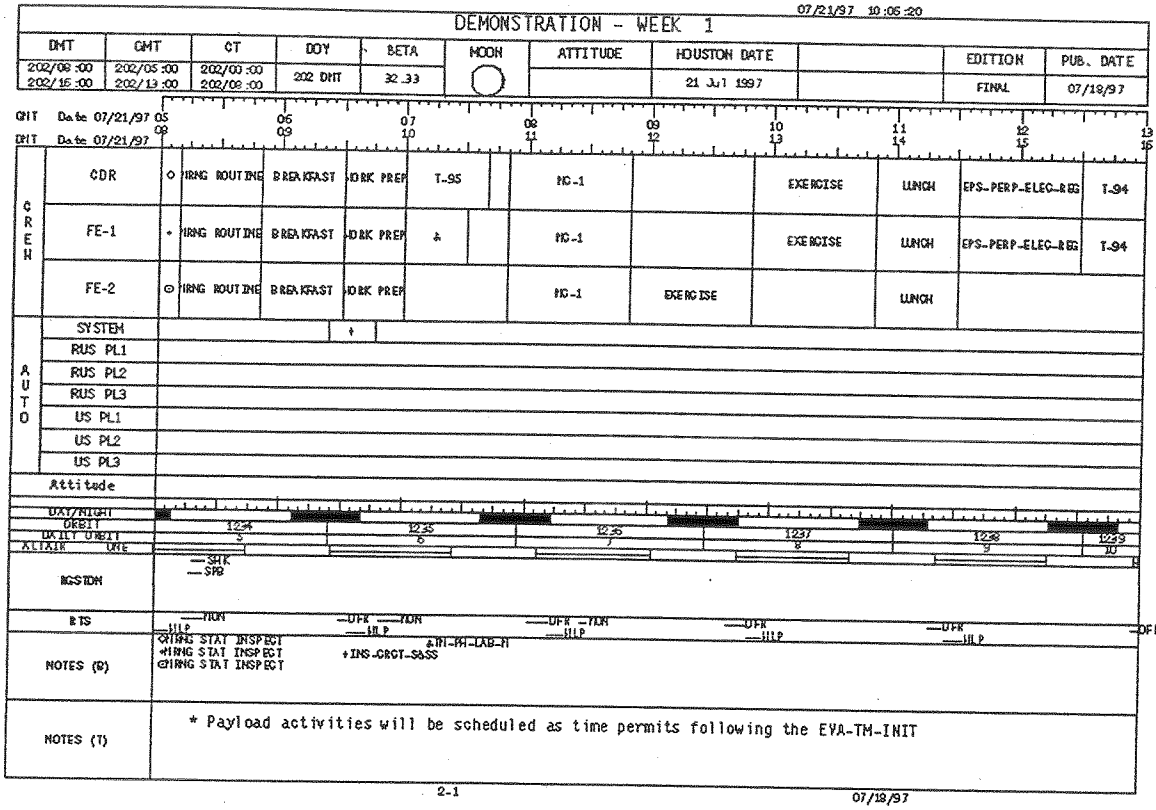
Ultimately, the Phase I Demonstration was a success. The three centers were able to successfully developed two 1-week Short Term Plans for the Mir long duration mission using CPS/IFAP. The final product proved that the artificial intelligence has the capability to schedule and plan complex timelines for the cosmonauts and astronauts onboard the Mir Space Station and for International Space Station.

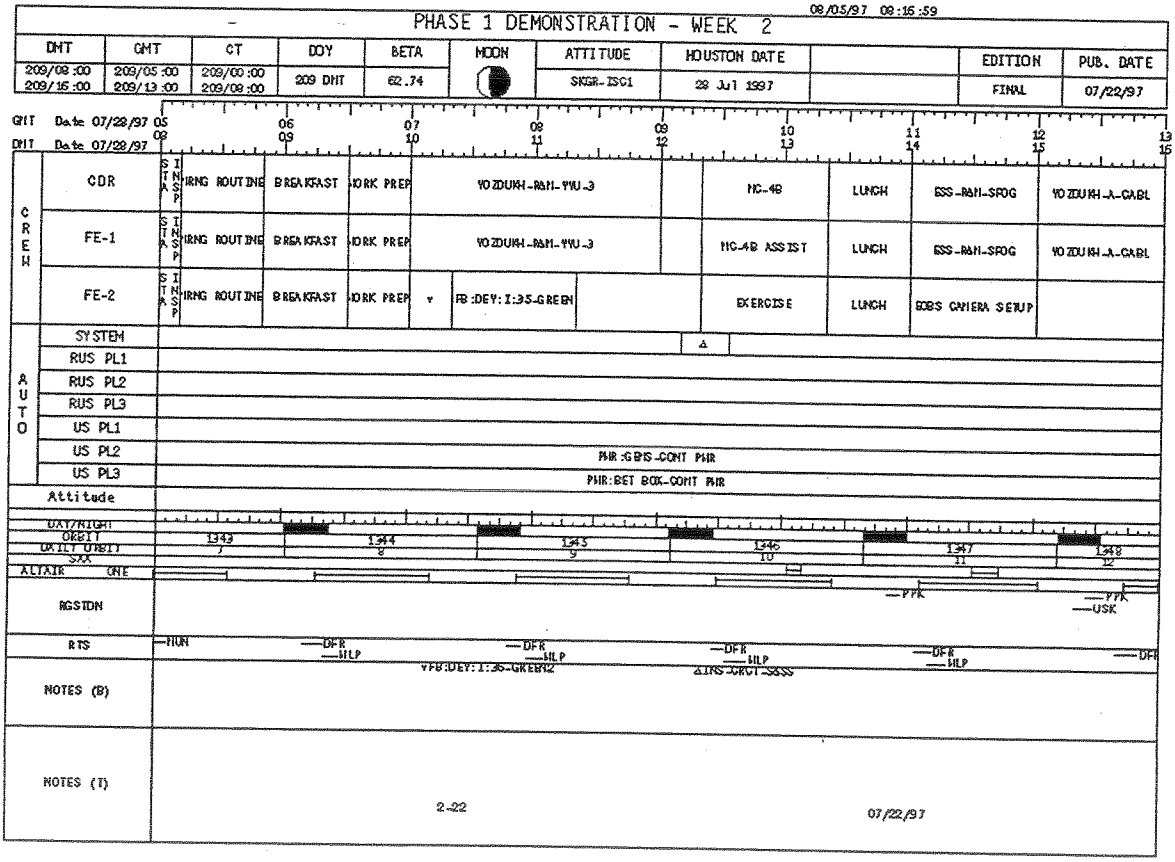
Despite extensive problems onboard the Mir, planners were able to modify their process to meet delivery deadlines. Other major accomplishments of the Demonstration include:

- Planners gained invaluable experience in planning systems operations and responding to off-nominal situations under adverse conditions;
- Russian planners were trained on using the CPS/IFAP tool and showed proficiency on the system;
- Planners established together a detailed list of joint procedures, which could be used as the basis for early increments;
- Database standards and activity parameters were documented and agreed to;

- Extensive platform and software problems were found and recommendations were submitted to the Development community.
- Planners established extensive working relationships.

Many of the planners who worked with the Phase I Demonstration have gone on to use these experiences as the foundation of planning for the NASA/Mir Long Duration Missions (LDM). CPS/IFAP is currently being used to plan and schedule the US science activities for LDM 6 and 7. Where American astronauts, Dave Wolf and Andy Thomas, respectively, will be onboard the Mir Station. JSC planners will then use all the information that was learned during both remaining LDMs and apply it to planning for the first crew's activities on-board the International Space Station.


07/21/97 10:06:20
DEMONSTRATION - WEEK 1
DMT | GMT | CT | DOY | BETA | MOON | ATTITUDE | HOUSTON DATE | EDITION | PUB. DATE
202/08:00 202/05:00 202/00:00 202 DMT 32.33 21 Jul 1997 FINAL 07/18/97
202/16:00 202/13:00 202/08:00
GMT Date 07/21/97
DMT Date 07/21/97
CREW
CDR
FE-1
FE-2
AUTO
SYSTEM
RUS PL1
RUS PL2
RUS PL3
US PL1
US PL2
US PL3
Attitude
DAY/NIGHT
ORBIT
DAYLT ORBIT
ALTAIR ONE
RGSTDN
RTS
NOTES (R)
NOTES (T)
* Payload activities will be scheduled as time permits following the EVA-TM-INIT
2-1
07/18/97


08/05/97 08:16:59
PHASE 1 DEMONSTRATION - WEEK 2
DMT | GMT | CT | DOY | BETA | MOON | ATTITUDE | HOUSTON DATE | EDITION | PUB. DATE
209/08:00 209/05:00 209/00:00 209 DMT 62.74 SKGR-ISC1 28 Jul 1997 FINAL 07/22/97
209/16:00 209/13:00 209/08:00
GMT Date 07/28/97
DMT Date 07/28/97
CREW
CDR
FE-1
FE-2
AUTO
SYSTEM
RUS PL1
RUS PL2
RUS PL3
US PL1
US PL2
US PL3
PWR:GBPS-CONT PWR
PWR:BET BOX-CONT PWR
Attitude
DAY/NIGHT
ORBIT
DAYLT ORBIT
SAA
ALTAIR ONE
RGSTDN
RTS
NOTES (B)
NOTES (T)
2-22
07/22/97

References

Enticknap, S. 1997. http;//Shuttle-mir.nasa.gov.

Grethen, M. 1996. Increment Operations Plan. http://iss-www.jsc.nasa.gov/ss/issapt/iop/homepage.html.

Korth, D.; Reed, T.; Stanilovskaya, V. 1997. Phase I Demonstation Final Report. Forthcoming

Lubinsky, V., Shinkle, J., and Weiler, J. 1997. Phase I Demonstration Implementation Plan.

NASA/Flight Planning & Tool Development Group. 1996. CPS Training Manual.

Stottler Henke Assoc.,Inc. 1997. Intelligent Flight Activities Planner, Version 4.0.

Program Action Plan Automation Tool: Scheduling Large Numbers of Space Assets for the 3rd Space Operations Squadron

James N. Ortiz, Ph.D
NASA Exchange Engineer
Phillips Research Complex
Air Force Research Laboratory
Kirtland AFB, New Mexico
ortizj@plk.af.mil

Janet Wallace
3rd Space Operations Squadron Support
LTMO Space Operations Support Contract
Falcon AFB, Colorado
wallacjl@fafb.af.mil

Abstract

The U.S. Air Force satellite resource scheduling problem involves the daily allocation of Air Force Satellite Control Network (AFSCN) resources. These resources consist of tracking stations located all over the world, that serve hundreds of support requests each day for telemetry and commanding services from defense satellites orbiting the earth. The Air Force Research Laboratory's Phillips Research Complex is collaborating with the 3rd Space Operations Squadron (3rd SOPS) to conduct research in technologies that provide increased automation of work-intensive satellite scheduling functions, and is implementing these systems on low cost Personal Computer (PC) platforms. This paper describes the development, implementation, and fielding of a tool for the preparation and submission of support requests for AFSCN services by the 3rd SOPS. This paper contains a discussion of the lessons learned while fielding the tool, as well as a discussion of future research efforts that could augment the tool's capabilities in areas such as multi-user collaborative scheduling and automated optimal scheduling.

Introduction

The 3rd Space Operations Squadron (3rd SOPS) operates several families of defense communications satellites. The 3rd SOPS uses the AFSCN to perform health and status monitoring, tracking, and commanding of these satellites. As one of the many users of the AFSCN, 3rd SOPS must submit a Program Action Plan (PAP) for each satellite every week to the AFSCN schedulers. PAPs are used to specify support windows, support criteria (such as concurrent supports and maximum time between supports), and late starts and early stops (Abbot 1996). PAPs maybe also be used to define support preferences such as the required antenna side or unacceptable equipment. PAPs are submitted two weeks prior to the week of the requested support times. It is at this level that the scheduling process for the AFSCN begins, and for this reason it is very important for the efficiency of the overall scheduling process that PAPs be highly accurate and correct. Each satellite has specific requirements for the types of supports needed, the length of those supports, the frequency of each type of support, tolerances for how much the support can be moved from its desired time, and constraints for total resource utilization. Preparing PAPs is very labor intensive and is done manually for the most part. The objective of the PAP automation tool described in this paper is to reduce the effort required to build a PAP and to more efficiently prevent conflicts between support requests from various vehicles in the operations center.

Problem Domain

The 3rd Space Operations Squadron (3rd SOPS) controls several families of defense communications satellites. The satellites are operated out of two Space Operations Complexes (SOCs): a primary and a backup. Every week, the mission schedulers write a Program Action Plan (PAP) that covers a one-week period beginning two weeks in the future. This PAP includes all the supports required for the week in question. Mission (or support) requirements are dictated by the Orbital Operations Handbook (OOH) and the Orbital Requirements Document (ORD). Mission requirements are broken down into three categories: special, routine, and baseline. Special requirements are those that have no set schedule for when they need to be accomplished. They are scheduled whenever the vehicle engineers and orbital analysts determine that a special support is needed. By contrast, routine supports have a set schedule frequency, and baseline supports such as the state of health checks and tracking supports, must be run three times a day. The 3rd SOPS runs a minimum of 325

supports a week just for baseline supports. Whenever possible, routine and special supports are combined with a baseline support to keep the number of supports down. The 3rd SOPS primary SOC can run a maximum of six simultaneous supports. The backup SOC is only used when the primary is not functional. For example, when routine maintenance must be performed on the primary SOC, no supports can be run from this location. The backup SOC is used in situations such as this. Therefore, part of the scheduling process involves moving supports around scheduled downtime, and limiting the number of simultaneous contacts while still meeting mission requirements. The goal of this program is to automate as much of the process as possible.

Automation Tool

The PAP automation tool is a database driven computer program that provides the functionality of a "smart assistant" that guides and assists the user in the preparation of the PAP.

The functionality for creating a PAP is broken down into three areas: mission requirements, support requests, and conflict avoidance. Mission requirements are operational activities that are carried out through support requests. These requirements are of different types such as special, baseline, routine, or operations center requirements. Specific information for each type of mission requirement is stored in records in a database. Mission planners build support requests with the "click of a button." The information contained in the requirement records tells the program how to create the support requests (Figure 1).

After all of the support requests are built, the mission planners run a conflict avoidance routine that checks for scheduling conflicts such as supports scheduled during system downtime or violations in pre-specified limits. This conflict avoidance routine attempts to perform its task automatically by moving the support requests within the operations constraints. Whenever the tool is unable to resolve a conflict, the user is notified that human intervention is needed.

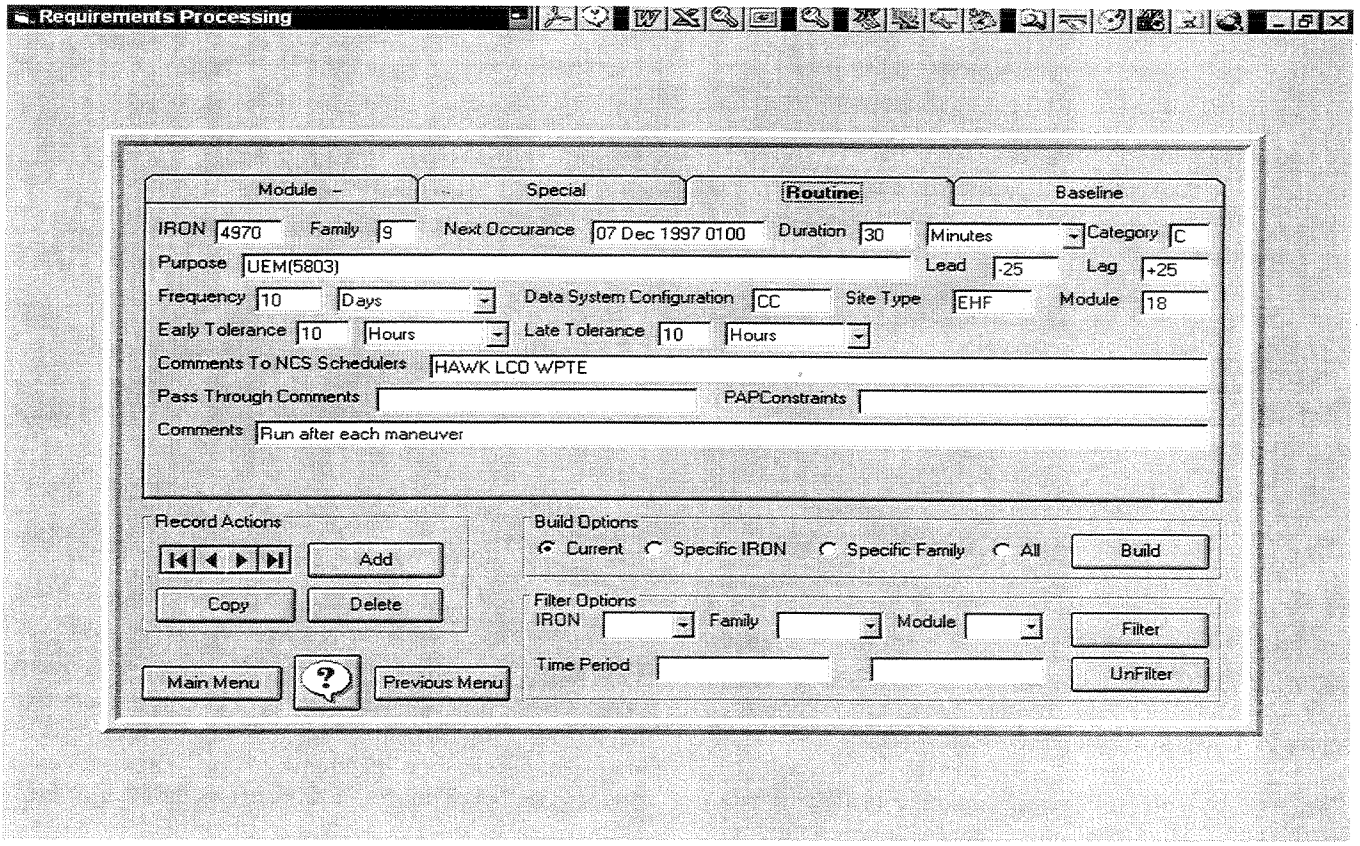


Figure 1. Support entry screen

The user has two ways to view and edit the support requests on the screen: as a form that displays one record at a time, and as a graphical display.

Graphical Display

3rd SOPS operators eventually transcribe the PAP information into a drawing that depicts all the support requests on a timeline. The information is manually entered into hardcopy templates that represented the PAP scheduling period.

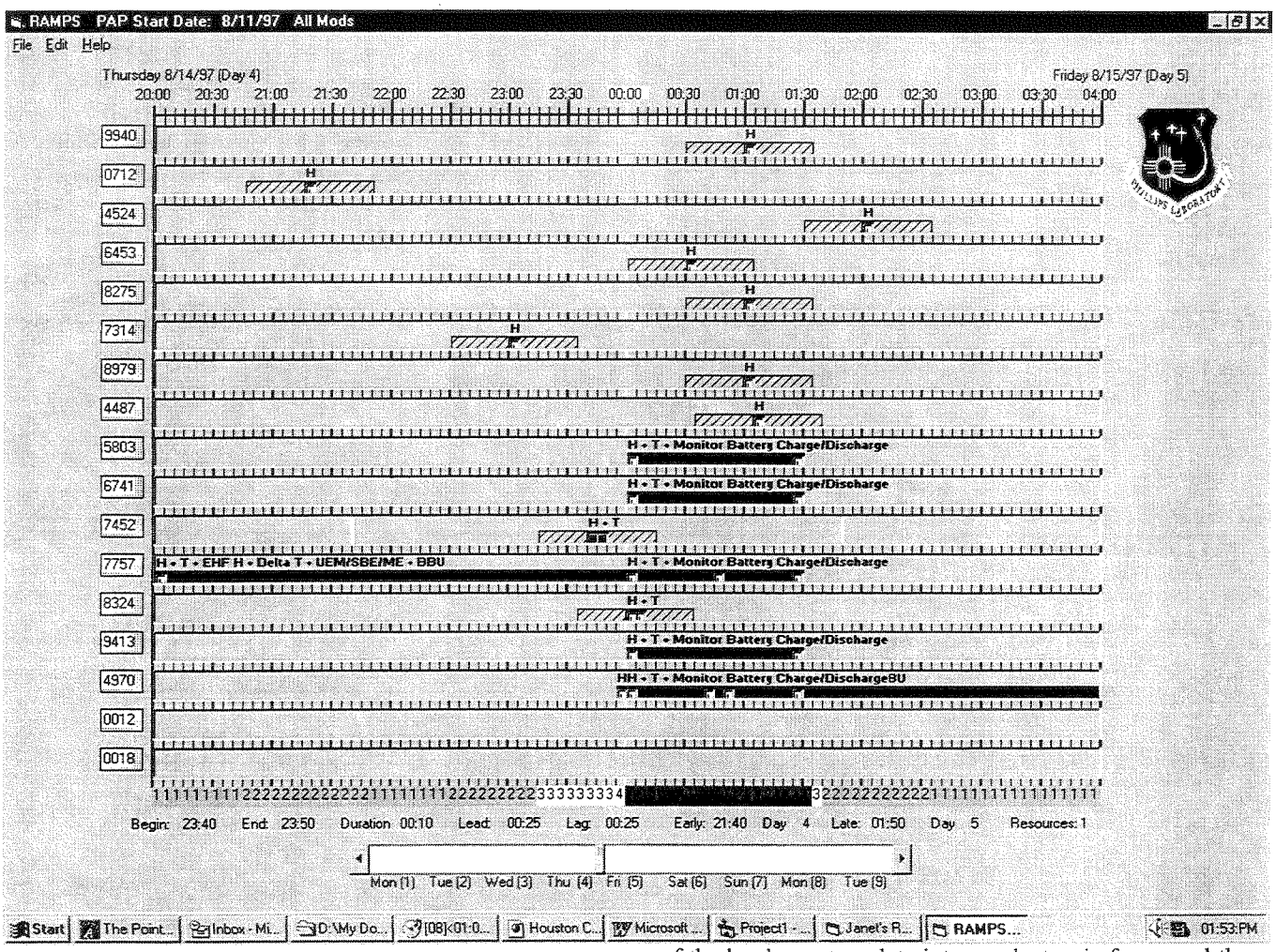


Figure 2. Graphical component main screen

Each support request is drawn as a box on the appropriate satellite's timeline, and other markings are added to represent information related to the scheduling window for the support. Labels are also added to identify the purpose of the support. These makings have specific formats that had been standardized over the years. Operators use these drawings to visualize and manually perform conflict resolution on the PAP. Conflict resolution consists of preventing overlaps between supports as well as balancing the total number of control room resources required for concurrent support to several satellites. Any changes to the information in the PAP that resulted from the conflict resolution process are manually edited on the drawing. Eventually, when all the conflicts are resolved, this drawing is turned over to the AFSCN resource scheduling squadron for further scheduling.

The approach taken on the development of the graphical display (Figure 2) was to start by building the functionality of the hardcopy template into an electronic form, and then enhancing it to include additional capabilities such as increased visualization, electronic editing, automated error checking, resource utilization advice, and online user help. This approach had the advantage of providing the user with a graphical tool that looks very similar to the existing template, thus minimizing the learning time and effort required to use it. The small amount of learning time that was needed was primarily devoted to learning the new capabilities included with the graphical display. An

additional advantage of this approach is that it complies with the standard notation being used to communicate with the AFSCN resource scheduling squadron.

The graphical user interface consists of a main screen, and several support screens used to solicit inputs from the operator. The main screen is used by the operator to visualize and edit the information on the PAP (Figure 3). To start a session, the graphical component first provides menus for specifying the desired scheduling period, and then proceeds to search the database for the requested scheduling information. This information is displayed in a group of scheduling timelines corresponding to every satellite operated by the Space Operations Complex (SOC). Nine days of scheduling information are presented to the user: the seven-day PAP period plus two extra days. The main screen displays this information in eight-hour blocks, with a scroll bar control provided to move across the total nine-day scheduling period. The timelines are marked across the top with date and times corresponding to the eight-hour block being displayed.

All support requests for a given satellite are displayed along its timeline as scheduling boxes. These scheduling boxes are marked with all the relevant information for that support, such as its purpose, number of resources, lead and lag times, and early and late times. These markings are consistent with the standard PAP notation, such as crosshatched boxes for the lead times and lag times, and solid lines for the earlier and later times. This information is also presented in numerical format in readouts that are displayed when the user selects a scheduling box. The user can edit the scheduling requests on the screen by dragging the scheduling boxes in the timeline to change its beginning and end times. Existing supports can be deleted and new supports can be added to any of the timelines. The user can also change the value of any parameter for any scheduling box, such as the early, late, lead, lag times, purpose label, or the number of resources. The graphical component displays all the timelines in parallel in order to provide the user with a comprehensive view of all the supports requested for a given time period. This overview facilitates the visualization and resolution of any potential conflicts. To aid the operator in the conflict resolution task, a tally of all the resources requested for every five-minute scheduling period is displayed below the group of timelines. These "resource-count" fields are highlighted in three different colors: green, yellow or red, corresponding to the normal, warning, or caution resource usage ranges. Operators use this information to reposition the supports in the timelines such that the resource usage does not exceed the resource capacity of the operations room. The order in which the timelines are displayed is also controlled by the operator to allow close comparison between the individual supports for any combination of satellites.

The graphical component also implements automated error checking capabilities. The operator is prevented from positioning support requests outside the scheduling windows defined by the earlier and late times. Also, error checking is included within every user-input screen to assure that the information entered by the operator is allowable. When invalid information is entered, a dialog window is displayed with a message indicating the discrepancy and how the operator may correct it.

An extensive set of on-line user help is included with the graphical component. This user help provides guidance to the operator on the use of the tool and all its different features. The user help is displayed in the standard help format used by Microsoft applications.

Software Tools and Integration

Two different versions of Microsoft Visual Basic were used to develop the PAP automation tool. Visual Basic 4.0 was used for all development work, with the exception of the graphical component, which was developed using Visual Basic 5.0. The graphical component and the rest of the PAP automation tool were developed independently by the Air Force Research Laboratory and the 3rd Space Operations Support Contractor, respectively. A Microsoft Access database was used to store all the PAP data. The graphical tool component communicates with the rest of the PAP automation tool via the Access database. The integration of the graphical component into the PAP automation tool software was accomplished by implementing the graphical component as a reusable software component or an Active X control. Visual Basic 5.0 was used to turn the graphical component application into an Active X control. The Active X control was then added to the rest of the PAP automation as a custom control within Visual Basic 4.0 (Figure 3). The integration of the Active X control into the rest of the software was very successful and uneventful. The tool was developed in a relatively short period of time (about four months) and runs on personal computers. The tool has greatly decreased the amount of time it takes the mission planners to build and perform conflict resolution on the PAP. What used to take 10 to 12 hours a week can now be done in half the time. (These time estimates are based on data gathered during the testing of the program.)

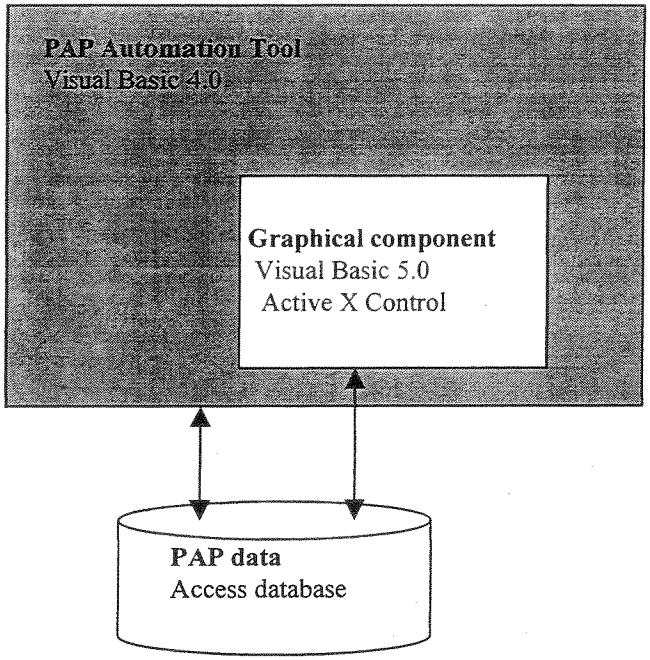


Figure 3. PAP automation software components

Deployment

An initial version of the PAP automation tool was delivered in April 1997. At this point the mission planners began testing the program and returning comments. Some of the comments were for new requirements, while others pointed out minor problems with the code. In the summer of 1997, fixes and new requirements were added to the tool. During this time, the Air Force Research Laboratory developed the graphical control for the tool. This was delivered in August 1997 to the contractor. 3rd SOPS operators are continuing to test the tool and request changes. According to the current timeline, 3rd SOPS will begin operations with the tool in October 1997.

Relevance to Non-DOD Missions

Even though this tool was designed specifically to address the scheduling request problem of the Air Force Satellite Control Network, it can also be applied to other problem situations requiring scheduling large numbers of space assets, a problem faced by other space organizations, in both the public and the private sectors. The large number of satellite constellations that are planned to go on orbit in the next few years will pose a very challenging problem in this area. DOD has extensive experience operating large satellite networks. Tools such as the PAP automation tool benefit from this experience by building on existing proven processes used by the DOD to operate these large networks, and improving these processes by increasing their level of automation.

Future Work

Future work on this area will investigate the possibility of adding a collaborative scheduling request capability. This capability would allow operators located at different locations to work on the same PAP, at the same time. Our research will answer a variety of issues related to how the latest scheduling information is broadcast and what levels and types of permissions are granted to the different users to edit each other's data when conflicting requests arise. Another area of further research will consist of adding an automated scheduling capability that optimally produces the AFSCN schedule (Ortiz 1997). Lastly, the addition of an embedded training capability is being investigated. This capability would be implemented as another software component that will be completely integrated into the tool's software and would provide training to the user by utilizing the tool's displays and controls.

Conclusion

This paper describes the development, implementation and fielding of a tool to automate the preparation and submission of support request for AFSCN services. The tool was developed in a relative short period of time and runs on inexpensive computer platforms. The tool had a very large impact on the way scheduling request operations are performed by the 3rd Space Operations Squadron, at Falcon AFB. This project also provided an example of successful collaboration between an operations organization, its support contractor, and a research laboratory.

References

Abbot, R.J., et al. (1996). Automated Scheduling in the Satellite Control Network. *Technical Report, The AeroSpace Corporation,* El Segundo, California.

Ortiz, James N. (1997). A Discrete Event Systems Approach to Automating Resource Scheduling on the USAF Satellite Network. *In Proceedings of the 2nd International Symposium on Reducing the Cost of Spacecraft Ground Systems and Operations.*

Robust Periodic Planning and Execution for Autonomous Spacecraft *

Barney Pell † Erann Gat § Ron Keesing † Nicola Muscettola ‡ Ben Smith §

Abstract

The New Millennium Remote Agent (NMRA) will be the first on-board AI system to control an actual spacecraft. The spacecraft domain raises a number of challenges for planning and execution, ranging from extended agency and long-term planning to dynamic recoveries and robust concurrent execution, all in the presence of tight real-time deadlines, changing goals, scarce resource constraints, and a wide variety of possible failures. NMRA is one of the first systems to integrate closed-loop planning and execution of concurrent temporal plans. It is also the first autonomous system that will be able to achieve a sustained, multi-stage, multi-year mission without communication or guidance from earth.

1 Introduction

We are developing the first on-board AI system to control an actual spacecraft. The mission, Deep Space One (DS-1), is the first in NASA's New Millennium Program (NMP), an aggressive series of technology demonstrations intended to push Space Exploration into the 21st century. DS-1 will launch in mid-1998 and will navigate and fly by asteroids and comets, taking pictures and sending back information to scientists on Earth. One key technology to be demonstrated is spacecraft autonomy, including on-board planning and plan execution. The spacecraft will spend long periods without the possibility of communication with ground operations staff and will, in fact, plan how and when it will communicate back to Earth. It must maintain its safety and achieve high-level goals, when possible, even in the presence of hardware faults and other unexpected events.

This paper describes our approach to planning and plan execution in the context of spacecraft autonomy. Our approach is being implemented as part of the New Millennium Remote Agent (NMRA) architecture [Pell *et al.*, 1997]. This architecture integrates traditional real-time monitoring and control with constraint-based planning and scheduling [Muscettola, 1994], robust multi-threaded execution [Gat, 1996], and model-based diagnosis and reconfiguration [Williams & Nayak, 1996].

The paper is organized as follows. Section 2 discusses the spacecraft domain and requirements which influence our design. Section 3 describes our approach to planning, execution, and robustness, and illustrates the top-level loop of our system. Section 4 addresses the issues involved in generating plans to support robust execution, and Section 5 shows how such plans are executed. We then consider related work and conclude.

2 Domain and Requirements

The autonomous spacecraft domain presents a number of challenges for planning and plan execution. Many devices and systems must be controlled, leading to multiple threads of complex activity. These concurrent processes must be coordinated to control for negative interactions, such as vibrations of the thruster system violating stability requirements of the camera. Also, activities may have precise real-time constraints, such as taking a picture of an asteroid during a narrow window of observability.

Virtually all resources on spacecraft are limited and carefully budgeted. The system must ensure that they are allocated effectively to goal-achievement. Some resources, like solar panel-generated power, are renewable but limited. Others, such as total propellant, are finite and must be budgeted across the entire mission. The planner reasons about resource usage in generating plans, but because of run-time uncertainty the resource constraints must also be enforced as part of execution.

The planner and plan execution system must reason

*This paper appears in the Proceedings of IJCAI-97.

‡Recom Technologies, NASA Ames Research Center, MS 269/2, Moffett Field, CA 94035.

†Caelum Research, NASA Ames Research Center, MS 269/2, Moffett Field, CA 94035.

§Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91109.

about and interact with external agents and processes, such as the on-board navigation system and the attitude controller. These external agents can provide some information at plan time and can achieve tasks and provide more information at run-time but are never fully controllable or predictable. For example, the attitude controller can provide estimates of turn durations at plan time, but the completion of turns during execution is not controllable and can only be observed. Plans must express compatibilities among activities, and the plan execution system must synchronize these activities at run-time.

In addition, planning and the information necessary to generate plans are also limited resources. Because of the limited on-board processing capabilities of the spacecraft, the planner must share the CPU with other critical computation tasks such as the execution engine, the real-time control loops and the fault detection, isolation and recovery system. While the planner generates a plan, the spacecraft must continue to operate. Moreover, the plan often contains critical tasks whose execution cannot be interrupted in order to install newly generated plans. Thus inserting a planning activity means many other activities must be suspended or postponed. Since planning is expensive, plan failure is costly. Thus plan execution must be robust in the face of a wide variety of hardware faults and delays.

3 Approach

Our approach separates an extensive, deliberative planning phase from the reactive execution phase, executing infrequently generated plans over extended time periods. How frequently and how far in advance the system should plan is constrained by several factors, including uncertainty about the results of execution. For example, uncertainty about how much thrust has accumulated after a thrusting maneuver means the system can't reliably plan how many thrusts will be needed to reach the target, thus reducing how far in advance it is productive to plan such thrusting activities. While uncertainty motivates frequent planning with short *scheduling horizons*, the cost of planning motivates plans with long horizons.

In our approach the generation of the next plan is explicitly represented in the current plan as a task. By considering the costs and constraints of planning, the planner automatically optimizes future planning activities. When the executive reaches this task in the current scheduling horizon, it asks the planner to generate a plan for the next scheduling horizon while it continues to execute the activities remaining in the current plan. When the executive reaches the end of the current horizon, the plan for the next horizon will be ready and the executive will then install it and continue execution seamlessly.

Ideally, we would like to have the planner represent the spacecraft at the same level of detail as the executive. This approach is taken by [Bresina *et al.*, 1996], and by [Levinson, 1994]. The approach, when feasible, has a number of benefits. First, it enables the planner to simulate the detailed functioning of the executive under various conditions of uncertainty, and to produce a plan which has contingencies (branches) providing quick responses for important execution outcomes. Second, it enables the use of one language rather than two for expressing action knowledge, which simplifies knowledge engineering and helps maintain consistency of interfaces. Third, it enables the planner to monitor execution in progress and project the likely course of actions, and then provide plan refinements which can be patched directly into the currently executing plan.

Unfortunately, in our domain this single representation approach is not practical because the complexity of interactions at the detailed level of execution would make planning combinatorially intractable. Thus, we have found it necessary to make the planner operate on a more abstract model of the domain. Examples of abstractions are:

- hiding details of subsystem interactions controlled by the executive
- merging a set of detailed component states into abstract states
- not modeling certain subsystems
- using conservative resource and timing estimates

These simplifications have several consequences which impact our design. One important consequence is that the planner can no longer model or predict intermediate execution states. Since the executive is managing multiple concurrent activities at a level of detail below the planner's visibility, it is difficult to provide a well-defined initial state as input to the infrequent planning process. Also, a newly generated plan may be invalidated by execution activities that occur during or even after planning but prior to execution of the new plan, since the initial conditions of that plan may no longer be consistent with the state of the spacecraft.

We address the problem of generating initial states for the next planning round differently depending on the status of the currently-executing plan. Plans normally include an activity to plan for the next horizon. At this point, the executive sends to the planner the current plan in its entirety, with annotations for the decisions that were made so far in executing it. The current plan serves as its own prediction of the future at the level of abstraction required by the planner. Thus, all the planner has to do is extend the plan to address the goals of the next planning horizon and return the result to the executive. The executive must then merge the extended plan with its current representation of the existing plan.

The net result is that, from the executive's perspective, executing multiple chained plans is virtually the same as executing one long plan. This has the useful consequence that it enables the executive to engage in activities which span multiple planning horizons (such as a 3-month long engine burn) without interrupting them.

In the event of plan failure, the executive knows how to enter a stable state (called a standby mode) prior to invoking the planner, from which it generates a description of the resulting state in the abstract language understood by the planner. Note that establishing standby modes following plan failure is a costly activity, as it causes us to interrupt the ongoing planned activities and lose important opportunities. For example, a plan failure causing us to enter standby mode during the comet encounter would cause loss of all the encounter science, as there is no time to re-plan before the comet is out of sight. Such concerns motivate a strong desire for plan robustness, in which the plans contain enough flexibility, and the executive has the capability, to continue executing the plan under a wide range of execution outcomes.

4 Planning

A principal goal of the NMRA is to enable a new generation of spacecraft that can carry out complete, nominal missions without any communication from ground. This is a great departure from previous and current missions (such as Voyager, Galileo or Cassini) which rely on frequent and extensive communications from ground. In traditional missions, ground operations routinely uplink detailed command sequences to be executed during subsequent mission phases. Such communications require costly resources such as the Deep Space Network, which makes them very expensive. Uplink independence is particularly important for missions that require fast reaction times (as it is the case for autonomous rovers, comet landers and other remote explorers); in this case detailed ground-based control is infeasible due to long communication lags. In case of loss of uplink capabilities, previous spacecraft could carry out a critical sequence of commands stored on board before launch. However, these sequences were greatly simplified when compared to the uplinked sequences and could only carry out a small fraction of all mission goals.

Mission Manager

NMRA is launched with a pre-defined "mission profile" that contains a list of all nominal goals to be achieved during the mission. The detailed sequence of commands to achieve such goals, however, is not pre-stored but is generated on board by the planner. A special module of the planner, the Mission Manager, determines the goals that need to be achieved in the next scheduling horizon (typically 2 weeks long), extracts them from the mission profile and combines them with the initial spacecraft state as determined by the executive. The result is a specific planning problem that, once solved, yields detailed execution commands. This decomposition into long-range mission planning and shorter-term detailed planning enables NMRA to undertake an extended diverse mission with minimal human intervention.

Requirements for Robust Execution

The NMRA must be able to respond to unexpected events during plan execution without having to plan the response. Although it is sometimes necessary to re-plan, this should not be the only option. Many situations require responses that cannot be made quickly enough if the NMRA has to plan them.

The executive must be able to react to events in such a way that the rest of the plan is still valid. To support this, the plan must be flexible enough to tolerate both unexpected events and the executive's responses without breaking. This flexibility is achieved by (1) choosing an appropriate level of abstraction for the activities and (2) generating plans in which the activities have flexible start and end times.

The abstraction level of the activities in the plan must be chosen carefully. If the activities are at too fine a level of granularity, then the plan will impose too many constraints on the behavior of the executive, making plan execution more fragile. However, if the granularity is too coarse, then there may be interactions among the sub-actions of activities that the planner cannot reason about. In DS1, activities are abstracted to the level where there are no interactions among their sub-activities. This level allows the planner to resolve all of the global interactions without getting into details that would over-constrain the executive.

The other mechanism by which the executive can respond to events without breaking the plan is having activities with flexible start and end times. Plans in DS1 consist of temporal sequences of activities. Each activity has an earliest start time, a latest start time, an earliest end time, and latest end time. The planner uses a least commitment approach, constricting the start and end times only when absolutely necessary. Any flexibility remaining at the end of planning is retained in the plan. This flexibility is used by the executive to adjust the start and end times of activities as needed. For example, if the engine does not start on the first try, the executive can try a few more times. To make time for these extra attempts, the end time is moved ahead, but not beyond the latest end time.

Changing the start or end time of an activity may also affect other activities in the plan. For example, if the spacecraft must take science data five minutes after shut-

ting down the engine, then changing the end time of the `engine firing` activity will change the start time of the `take science data` activity. To make the changes, the executive must know about the temporal constraint between the `fire engine` activity and the `take science data` activity. The plan therefore contains all of the temporal constraints among the activities.

Although the planner is typically *enabled* to leave flexibility in the activity start and end times because the times are under-constrained, it is sometimes *required* to provide such flexibility in order to operate the spacecraft successfully. For example, when the engine is commanded to turn on, it goes through a warm up procedure and turns itself on. The warm up procedure can take up to ten minutes, but the actual warm up time is not known at plan time. It is not known until the engine actually turns on. We currently handle such cases by providing enough time in activities to handle worst-case outcomes, although we are developing a method to plan explicitly about execution-time uncertainty.

5 Execution

From the point of view of the NMRA executive, a plan is a set of time-lines. Timelines consists of a linear sequence of tokens, each of which represents an activity which should be taking place during a defined temporal period. A token has a start and end window, a set of pre- and post-constraints. The start and end windows are intervals in absolute time during which the token must start and end. The pre- and post-constraints describe dependencies with respect to the starts and ends of tokens on other time-lines.

There are three different types of pre- and post-constraints: *before*, *after*, and *meets*. The semantics of these constraints is fairly straightforward. A before constraint specifies that the start of a token must come before the start of another token. An after constraint specifies that the end of a token must come after the end of another token. The amount of time that may elapse between these two related events is specified as an interval. A meets-constraint specifies that the start (end) of a token must coincide with the start (end) of another token.

Issues

Plan execution would be relatively straightforward were it not for the fact that different token types have different execution semantics. In particular, there are different ways of determining whether or not a particular activity has ended. Some activities are brought to an end by the physics of the environment or the control system (e.g. turns) while others are brought to an end simply by meeting all its internal plan constraints (e.g. the periods of constant-attitude pointing between turns).

The situation is further complicated by the fact that a naive operationalization of these constraints leads to deadlock. Consider a `constant-pointing` token A followed by a `turn` token B. Token A (waiting for the turn) should end whenever token B (the turn) is eligible to start. However, B is constrained by the planner to follow A, and so B is not eligible to start until A ends. Thus, A can never end, and B can never start.

Another issue is that some tokens don't achieve their intended post-conditions until some time after they have started. For example, consider a time-line for a device containing a token A of type `device-off` followed by token B of type `device-on`. The intent here is that the executive should turn the device on at the junction between A and B, but this cannot be done instantaneously. Thus, a token on another time-line, constrained to start after B, may fail if it depends upon the device being on, since the device may not in fact be turned on until some time after B starts. One possible solution to this problem is to change the planner model so that it generates a plan that includes an intermediate token of type `device-turning-on`, but this can significantly increase the size of the planner's search space, and hence the time and resources required to generate a plan.

To solve these problems, we separate the execution of a token into three stages: *startup*, *steady-state*, and *ending*. The startup stage performs actions to achieve the conditions that the planner intends the token to represent. The steady-state stage monitors and maintains these conditions (or signals failure if the conditions cannot be maintained). The ending stage allows the token to perform cleanup actions before releasing control to the next token on the time-line. Tokens may have null actions in one or more stages. The algorithm for executing a token in this three-phase framework is as follows:

1. Wait for the beginning of the token's start window.
2. In parallel
   (a) wait for token's pre-constraints to be true, and
   (b) check that the end of the start window has not passed. If it has, signal a failure.
3. Signal that the token has started.
4. Execute the achieve-portion of the token.
5. Spawn the maintain-portion of the token as a parallel task.
6. Wait for the start of the token's end window.
7. Wait for the token's post-conditions to be true.
8. Wait for the pre-conditions of the next token to be true, except those that refer to the end of this token.
9. Stop the maintain thread spawned in step 5, and execute the cleanup-portion of the token

10. Check that the end of the end window has not passed. If it has, signal a failure. Otherwise, signal that this token has ended.

This algorithm allows all the token types to be executed within a uniform framework.

6 Related Work

NMRA is one of the first systems to integrate closed-loop planning and execution of concurrent temporal plans. It is also the first autonomous system that will be able to achieve a sustained, multi-stage, multi-year mission without communication or guidance from earth.

Bresina *et al.* (1996) describe a temporal planner and executive for the autonomous telescope domain. Their approach uses a single action representation whereas ours uses an abstract planning language, but their plan representation shares with ours flexibility and uncertainty about start and finish times of activities. However, their approach is currently restricted to single resource domains with no concurrency.

Drabble (1993) describes the EXCALIBUR system, which performs closed-loop planning and execution using qualitative domain models to monitor plan execution and to generate predicted initial states for planning after execution failures. The "kitchen" domain involved concurrent temporal plans, although it was simplified and did not require robust reactions during execution.

Currie & Tate (1991) describe the O-Plan planning system, which when combined with a temporal scheduler can produce rich concurrent temporal plans. Reece & Tate (1994) developed an execution agent for this planner, and the combined system has been applied to many real-world problems including the military logistics domain. The plan repair mechanism [Drabble, Tate, & Dalton, 1996] is more sophisticated then ours, although the execution agent is weaker and does not perform execution-time task decomposition or robust execution.

The Cypress system [Wilkins *et al.*, 1995] and the 3T system [Bonasso *et al.*, 1996] also address the closed-loop integration of planning and execution in the context of concurrency, although neither of these systems deals with temporal plans. It is interesting to compare how these systems differ from ours concerning the generation of execution context for the planner and the integration of new planning information back into execution. Cypress shares the same action formalism between planning and execution. This enables the planner to watch over execution and simulate the results forward, as discussed in section 3. The planner can detect problems in advance and send back a detailed plan refinement, and the executive can replace un-executed portions of its current plan with new portions and continue running uninterrupted.

In 3T, the planner maintains such tight control over execution that it does not even send the full plan it has developed. Instead, it sends directives to the executive one at a time, and the executive then responds to each directive in turn. This provides an interesting solution to the problem of keeping the planner informed about execution and also to the problem of integrating new planning information into the execution context. However, this approach is problematic in our domain as it places severe time constraints on the planner so that it can decide what to do before the executive runs out of activities, and it requires the computational and informational resources to be available for planning on a continuous basis. This is a luxury we could not afford on a spacecraft, as discussed in section 3.

Other systems integrating planning and execution in real-world control systems include Guardian [Hayes-Roth, 1995], SOAR [Tambe *et al.*, 1995], Atlantis [Gat, 1992] and TCA [Simmons, 1990]. These systems invoke planning as a means to answer specific questions during execution (like whether a particular treatment would take effect in time to heal the patient, which evasive maneuver will counter the opponents current attack plan, and which path to take to get to a particular room). This use of planning contrasts with our approach, in which the planner coordinates the global activity in the system. The local approach has the advantage of making use of special-purpose planners which can be built to answer narrow questions, but our global approach has the advantage of ensuring that the different activities undertaken at execution will not interact harmfully. It is not clear how the local approaches can be extended to provide similar guarantees.

7 Conclusion

A growing body of work is addressing issues of robust planning and execution in the face of failures and uncertainty. The Lockheed Underwater Vehicle [Ogasawara, 1991] uses decision-theoretic planning and execution to select courses of action which maximize utility. CIRCA [Musliner, Durfee, & Shin, 1993] considers a set of states, actions, and critical failures to be avoided. It then inserts a set of sense-act transitions into a real-time controller to ensure that the controller will never enter the critical failure states. Cassandra [Pryor & Collins, 1996], Buridan [Draper, Hanks, & Weld, 1994], O-Plan [Currie & Tate, 1991] and JIC [Drummond, Bresina, & Swanson, 1994] all consider actions with uncertain outcomes and produce plans that enable execution-time recovery without having to take time out for replanning.

We are currently working on extending our planning approach to support such capabilities in the context of concurrent temporal plans. Our present levels of robustness are achieved using the complementary approach of

flexible, abstract, and conservative plans which can be exploited by a smart executive.

A final distinction between NMRA and most other planning and execution systems is that our planner actually plans how and when it will plan for the next horizon. That is, it inserts a "plan next horizon" activity into the plan and plans other supporting activities around this goal. Such activities include information-gathering activities which will be necessary before another plan can be built. The executive then achieves these activities to enable this form of planning over multiple horizons. We believe this is a necessary capability of extended agency, and one which will become of growing concern as we design autonomous agents to achieve goals unassisted over years or decades of activity.

References

[Bonasso *et al.*, 1996] Bonasso, R. P.; Kortenkamp, D.; Miller, D.; and Slack, M. 1996. Experiences with an architecture for intelligent, reactive agents. *JETAI.*

[Bresina *et al.*, 1996] Bresina, J.; Edgington, W.; Swanson, K.; and Drummond, M. 1996. Operational closed-loop obesrvation scheduling and execution. In Pryor [Pryor, 1996].

[Currie & Tate, 1991] Currie, K., and Tate, A. 1991. O-plan: the open planning architecture. *Artificial Intelligence* 52(1):49–86.

[Drabble, Tate, & Dalton, 1996] Drabble, B.; Tate, A.; and Dalton, J. 1996. O-plan project evaluation experiments and results. Oplan Technical Report ARPA-RL/O-Plan/TR/23 Version 1, AIAI.

[Drabble, 1993] Drabble, B. 1993. Excalibur: A program for planning and reasoning with processes. *Artificial Intelligence Journal* 62(1):1–40.

[Draper, Hanks, & Weld, 1994] Draper, D.; Hanks, S.; and Weld, D. 1994. Probabilistic planning with information gathering and contingent execution. In *Proceedings of AIPS94*, 31–36. AAAI Press.

[Drummond, Bresina, & Swanson, 1994] Drummond, M.; Bresina, J.; and Swanson, K. 1994. Just-in-case scheduling. In *Procs. of AAAI-94*, 1098–1104. Cambridge, Mass.: AAAI Press.

[Gat, 1992] Gat, E. 1992. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Procs. of AAAI-92.* Cambridge, Mass.: AAAI Press.

[Gat, 1996] Gat, E. 1996. ESL: A language for supporting robust plan execution in embedded autonomous agents. In Pryor [Pryor, 1996].

[Hayes-Roth, 1995] Hayes-Roth, B. 1995. An architecture for adaptive intelligent systems. *Artificial Intelligence* 72.

[Levinson, 1994] Levinson, R. 1994. A general programming language for unified planning and control. *Artificial Intelligence* 76.

[Muscettola, 1994] Muscettola, N. 1994. HSTS: Integrating planning and scheduling. In Fox, M., and Zweben, M., eds., *Intelligent Scheduling.* Morgan Kaufmann.

[Musliner, Durfee, & Shin, 1993] Musliner, D.; Durfee, E.; and Shin, K. 1993. Circa: A cooperative, intelligent, real-time control architecture. *IEEE Transactions on Systems, Man, and Cybernetics* 23(6).

[Ogasawara, 1991] Ogasawara, G. H. 1991. A distributed, decision-theoretic control system for a mobile robot. *ACM SIGART Bulletin* 2(4):140–145.

[Pell *et al.*, 1997] Pell, B.; Bernard, D. E.; Chien, S. A.; Gat, E.; Muscettola, N.; Nayak, P. P.; Wagner, M. D.; and Williams, B. C. 1997. An autonomous spacecraft agent prototype. In Johnson, W. L., ed., *Proceedings of the First Int'l Conference on Autonomous Agents.* ACM Press.

[Pryor & Collins, 1996] Pryor, L., and Collins, G. 1996. Planning for contingencies: A decision-based approach. *JAIR* 4:287–339.

[Pryor, 1996] Pryor, L., ed. 1996. *Proceedings of the AAAI Fall Symposium on Plan Execution.* AAAI Press.

[Reece & Tate, 1994] Reece, G., and Tate, A. 1994. Synthesizing protection monitors from causal structure. In *Procs. AIPS-94.* AAAI Press.

[Simmons, 1990] Simmons, R. 1990. An architecture for coordinating planning, sensing, and action. In *Procs. DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, 292–297. San Mateo, CA: DARPA.

[Tambe *et al.*, 1995] Tambe, M.; Johnson, W. L.; Jones, R. M.; Koss, F.; Laird, J. E.; Rosenbloom, P. S.; and Schwamb, K. 1995. Intelligent agents for interactive simulation environments. *AI Magazine* 16(1):15–39.

[Wilkins *et al.*, 1995] Wilkins, D. E.; Myers, K. L.; Lowrance, J. D.; and Wesley, L. P. 1995. Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical AI* 7(1):197–227.

[Williams & Nayak, 1996] Williams, B. C., and Nayak, P. P. 1996. A model-based approach to reactive self-configuring systems. In *Procs. of AAAI-96*, 971–978. Cambridge, Mass.: AAAI.

Automating Schedule Development for Shuttle Payload Operations

G. Rabideau[1], S. Chien[1], C. Eggemeyer[1], T. Mann[1], J. Willis[2], S. Siewerts[2], P. Stone[3]

[1]Jet Propulsion Laboratory
4800 Oak Grove Dr, MS 525-3660
Pasadena, CA 91109-8099
{gregg.rabideau, steve.chien,
tobias.mann, curt.eggemeyer}
@jpl.nasa.gov

[2]University of Colorado
Colorado Space Grant College
Campus Box 520
Boulder, CO 80309
{jason.willis,sam.siewert}
@colorado.edu

[3]Carnegie Mellon University
Computer Science Department
Pittsburgh, PA 15213-3891
pstone@cs.cmu.edu

Abstract

This paper describes the DATA-CHASER Automated Planner/Scheduler (DCAPS) system for automatically generating and repairing low-level command sequences for the DATA-CHASER shuttle payload. DCAPS uses general Artificial Intelligence (AI) techniques, including an iterative repair framework in which the system selectively resolves conflicts with the resource and temporal constraints of the payload activities.

Introduction

Command sequence generation for spacecraft operations can be a laborious process requiring a great deal of specialized knowledge. Command sets can be large, with each command performing a low-level task. There may be many interactions between the commands due to the use of resources. In addition, due to power and weight limitations, the resources available on-board spacecraft tend to be scarce. Because of this complexity, tools to assist in planning and scheduling spacecraft activities are critical to reducing the cost and effort of mission operations.

This paper describes a general system that uses Artificial Intelligence Planning and Scheduling technology to automatically generate command sequences for the DATA-CHASER shuttle payload operations. The DATA-CHASER Automated Planner/Scheduler (DCAPS) architecture presented supports direct, interactive commanding, rescheduling and repair, resource allocation, and constraint maintenance.

The DCAPS search algorithm was developed based on the "iterative repair" technique used in [1]. Basically, this technique iteratively selects a schedule conflict and performs some action in an attempt to resolve the conflict. Using a repair algorithm, DCAPS is naturally well-adapted for human interaction. Therefore, the scheduler can be used as a tool to assist payload command sequencing. With the use of this tool, sequencing becomes simple enough to be accomplished by nonspacecraft and sequencing experts, such as the mission scientists. This allows the scientist to become directly involved in the command sequencing process. Following any changes in spacecraft state or user-defined goals, the repair algorithm allows simple rescheduling that avoids disrupting the original schedule as much as possible. Finally, the highly restrictive payload resources and constraints are consistently monitored and conflicts avoided automatically.

The DCAPS system is being developed for operation of the DATA-CHASER shuttle payload, which is being developed and managed by students and faculty of the University of Colorado at Boulder. DATA-CHASER is a science payload, with a primary focus on solar observation. The main activities for the payload involve science instrument observations, data storage, communication, and control of the power subsystem. Science is performed using three solar observing instruments, Far Ultraviolet Spectrometer (FARUS), Soft X-ray and Extreme Ultraviolet Experiment (SXEE), and Lyman-alpha Solar Imaging Telescope (LASIT), that are imaging devices at various spectra.

The payload resources include power, tape storage, local memory, the three instruments, and the communication bus. DATA-CHASER is also constrained by externally-driven states such as the shuttle orientation, which affects when certain science activities can be scheduled. Payload activities must be sequenced while avoiding or resolving conflicts with resources and temporal constraints.

When using the DCAPS system, there are three modes of operation. First, by simply providing a small set of high-level science and engineering goals, an initial schedule can be generated. The goals, which describe high-level mission objectives, are automatically translated into a sequence of executable activities. The second phase offers an interactive scheduling session. Using the repair-based scheduler, the user can work with the low-level activities while maintaining consistency with resources and constraints.

After making any change in the schedule, the user can give one simple command to resolve all conflicts in the current schedule. A schedule free of conflicts, however, may not be the highest quality schedule. In the final stage, the user can call on the optimizer to generate several additional solutions based on preference information and select the best.

The main scheduling algorithm of the planner/scheduler is the repair-based search algorithm. Using this algorithm, the scheduler first collects all of the conflicts in the current schedule and classifies them based on the resource being violated and the culprit activities associated with the conflict. After choosing a conflict to repair, the scheduler

must select an action to perform in an attempt to resolve the conflict. Actions include moving, adding, and deleting activities. If the action resolves the conflict, the scheduler iterates on the resulting schedule. Otherwise, the scheduler tries a different action for resolving the persistent conflict.

The remainder of this paper is organized as follows. First, we describe the DATA-CHASER shuttle payload and mission objectives. Next, we describe how the payload is modeled. We then go into detail about the DCAPS approach to automated command sequence generation and repair. Then, we describe how DCAPS fits in to the overall flight and ground system architecture for the DATA-CHASER mission. Finally, we discuss related work and conclusions.

DATA-CHASER Payload

DATA-CHASER consists of two synergetic projects (see Figure 1), DATA and CHASER, which will fly as a Hitchhiker (HH) payload aboard STS-85 on the International Extreme Ultraviolet Hitchhiker Bridge (IEH-2) in July 1997 [2]. A technology experiment, DATA (Distribution and Automation Technology Advancement) seeks to advance semi-autonomous, supervisory operations. CHASER (Colorado Hitchhiker and Student Experiment of Solar Radiation) is a solar science experiment that serves to test DATA. The DATA technologies support cooperative operations distributed between different geographic sites as well as between humans and machines, on-board autonomy, human control, and ground automation.

CHASER is comprised of three coaligned instruments that take data in the far and extreme ultraviolet wavelengths. The first and oldest of these instruments (17 years old) is FARUS, which takes a continuous spectrum from 115 nm to 190 nm with a resolution of .12 nm. LASIT takes images of the full solar disk of the sun in the Lyman-alpha wavelength (121.6 nm) with a Charge Injected Device imager. The final instrument in the scientific package, SXEE, consists of four photometers, each having a different metallic coating so as to enable them to look at different wavelengths between 1 and 40 nm. The objective of these instruments is to measure the full disk solar ultraviolet irradiance and obtain images of the sun in the Lyman-alpha wavelength, providing a correlation between solar activity and radiation flux as well as an association of Lyman-alpha fluxes with individual active regions of the sun.

The flight segment of the DATA-CHASER project consists of a canister that is equipped with a Hitchhiker Motorized Door Assembly (HMDA), which houses the instruments and their support electronics. The second canister contains the flight computer for the payload as well as the 2 GB Digital Audio Tape (DAT) drive that is used to store all data that is collected during the mission. The payload data is also sent to the ground system through both low rate (available 90% of the time, at 1200 bps) and medium rate (available when scheduled, at 200 kbps). The payload is also capable of receiving commands sent from the ground system when uplink is available.

During the mission, the DATA-CHASER payload will be operating in four different modes. Most of the time, when DATA-CHASER is powered, it will be in a passive mode where it is monitoring its state and notifying the ground of any changes. During the time in the mission when the orbiter is scheduled to point the bay at the sun, the DATA-CHASER payload will shift into solar active mode where all instruments take data.

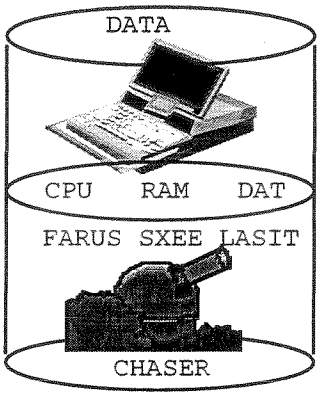


Figure 1: DATA-CHASER payload

The data is both written to the DAT drive on board and downlinked to the ground system for immediate data analysis. Several times during the mission, DATA-CHASER will take data while not pointing at the sun. This data is used for testing various portions of the DATA experiment with nonsolar pointing data in addition to being used for instrument calibration.

One of the consequences of flying on the shuttle system is that shuttle resources are limited, and their availability is subject to change every 12 hours. These resources include access to uplink and downlink channels, and time that your payload is allowed to operate. In addition to these resources, any given payload may also have environmental constraints as to how much contamination the payload can take. Another example is thermal constraints, such as maximum solar point time.

STS-85, the flight that DATA-CHASER payload is scheduled to fly on, is one of the most complicated flights that the shuttle has flown to date. In addition to the DATA-CHASER payload, there are four other payloads sharing the same HH bridge. In addition to the IEH-2 bridge, there is another HH bridge, a pallet payload, and a Spartan deployable satellite. Needless to say the shuttle pointing requirements are considerably tight.

In addition to modeling what the internal constraints and resources of the payload are, DCAPS must also search the shuttle flight plan for times when we are allowed to operate, downlink our data, uplink new command sets, and

when we have to protect the scientific instruments from contamination events.

DATA-CHASER is an interesting scenario for scheduling because of the complex data and power management involved in the science gathering. An automated scheduler must find an optimal "data taking" schedule, while adhering to the resource constraints. In addition, the scientists would like to perform dynamic scheduling during the mission. As an example, the summary data may indicate the presence of a solar flare. If this occurs, scientists have different requirements and goals, such as higher priorities on certain instruments or longer integration times. These new goals may require a different schedule of activities.

Modeling the Payload

In order to use the DCAPS system, the user must write a software model of the mission activities and spacecraft resources. This process involves defining a set of objects and how they interact. These definitions are then used by the scheduler to create instances of the objects. The two major types objects to be defined are activities and resources.

Activities

Activities are used to model the events that happen that affect the DATA-CHASER payload, and the actions that the DATA-CHASER payload can take. All activities have some basic components: a duration, a list of slots, and a list of slot-value assignments. The activity duration is simply a time range. Slots are parameters of activities that may represent resource usages. In addition, certain types of activities (described below) have a list of subactivities. For these activities, the user can also define a set of temporal constraints between the subactivities. Next, we describe in more detail the four basic types of activities: events, steps, step-activities, and activities.

Events are used to model activities that do not occur in a fixed relation to other activities (e.g. Tracking and Data Relay Satellite System (TDRSS) contacts) and are not part of an activity hierarchy.

Steps are the "leaf" nodes in the activity hierarchy tree. In other words, they do not contain any subactivities. Steps cannot be instantiated without their parents and are used to model the activities at the lowest level of detail. For instance, we model an activity called CHASER-heating, which consists of two steps, CHASER-heater-on and CHASER-heater-off.

Step-activities are used to model activities at a middle level of abstraction. They can contain steps, but must also have parent activities. In DCAPS, we model an activity SXEE-Data-Take, which models the SXEE instrument opening its aperture and taking a scan. In this case, there is a step-activity called SXEE-Scan-Step, which has sensor read steps and cannot be instantiated by itself.

Activities are used to model activities at the highest level of abstraction. They are the "root" nodes in the hierarchy tree, containing subactivities, but no parent activity. An abstract activity inherits all attributes of its predecessors. When it is detailed, the abstract activity is replaced by its subactivities, showing events and resource usages at a finer granularity. The activity and event objects are what the scheduler can instantiate, and methods are provided to access the varying levels of abstraction.

Resources

Resources define the various physical resources and the constraints they impose. Resources come in essentially five varieties: state, concurrency, depletable, nondepletable, and simple.

State resources are used to model the systems in the DATA-CHASER payload that have states associated with them. For each state resource, the modeler must specify the possible values that the state can be. Most of the systems have at least one state variable, which is whether or not they are activated. The orientation of the payload is also modeled as a state variable.

Concurrency resource constraints are used to model rules that stipulate that an activity either must occur with another activity or cannot occur with another activity. One relationship that is modeled with a concurrency resource is the requirement that a downlink or uplink can only occur during contact with a TDRSS satellite. This is modeled as a resource that is present when there is TDRSS contact activity, and required when there is a downlink or uplink activity.

Depletable resources are used to model resources with a fixed quantity, such a fuel or RAM. Activities can use some finite amount of a depletable resource, which may or may not be restorable. The amount used by the activity is persistent to the end of the schedule. In addition, the modeler must specify a maximum capacity for each depletable resource. In DCAPS, RAM is modeled as a depletable resource. Science observations produce data and use some amount of the depletable resource. Other activities, such as a transfer to permanent storage, may restore this resource.

Non-depletable resources are used to model resources which have a limit to the usage at any one time, but are reset at the end of the activity that consumes the resource. Similar to depletable resources, nondepletables are assigned a maximum capacity. Resources like power are modeled with nondepletable resources.

Simple resources are used to model devices that can only be used by one activity at a time. For instance, each of the instruments on board DATA-CHASER, FARUS, SXEE, and LASIT, are capable of taking only one image at a time and are modeled with simple resources.

Automated Planner/Scheduler

The DATA-CHASER Automated Planner / Scheduler will be part of the DATA-CHASER mission operations software. It will be a ground-based intelligent tool used for developing a schedule of commands for uplink to the payload. The user's manual [3] can be found at the Jet Propulsion Laboratory. There are three phases of operating the DCAPS system: a goal satisfaction phase, an interactive repair phase, and an optimization phase.

In the goal satisfaction phase, DCAPS produces a complete, valid schedule of payload operation commands from a model, initial state, and set of high-level goals. In the interactive repair phase, it takes intermediate, invalid schedules (resulting from user changes) and produce a similar, but valid schedule. Finally, in the optimization phase, the scheduler can take several valid schedules, score them, and select the most optimal schedule.

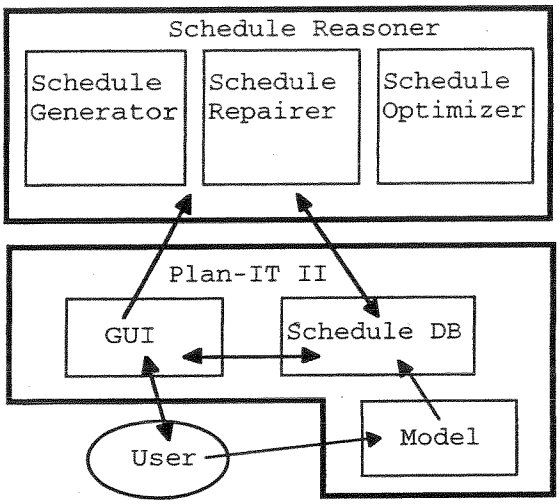


Figure 2: DCAPS architecture

The planner/scheduler consists of two main parts, the Plan-IT II (PI2) sequencing tool [4] and the schedule reasoner (see Figure 2). PI2 was written by William C. Eggemeyer and originally designed as an "expert assistant sequencing tool." PI2 includes a GUI that allows for easy manipulation of the schedule. In addition, it serves as an activity/resource database that supplies valuable information to the schedule reasoner. PI2 supports complex monitoring and reasoning about activities and the various constraints between them. The schedule reasoner uses Artificial Intelligence (AI) techniques to automatically generate new schedules, repair existing faulty schedules, and optimize valid schedules. PI2 provides information about resource availability and conflicts; the scheduler must decide which activities to use to resolve the conflicts and where to place the activities temporally. The two components work together to provide easy and fast sequencing of mission activities.

Schedule Data-Base

In the DCAPS system, PI2 is used primarily as a "schedule database" and resource constraint checker. It was originally developed as a graphical sequencing tool. Activities and resources are displayed on a graphical output. An activity represents some mission event that occurs over a period of time and uses some of the mission resources. A resources represents some limited available material whose usage is modeled as discrete blocks over time.

For each type of activity and resource, PI2 displays a timeline, which represents the behavior of that activity/resource type over a period of time. When activities are created, they are placed at a specified time on the timeline. Resources used by that activity are updated to reflect the additional usage. In addition to schedule visualization, PI2 provides an easy-to-use input interface for modifying the schedule. Moving activities is as simple as a click-and-drag with a mouse.

PI2 helps ease the burden on sequencers by continually monitoring all activities in the sequence. As activities are added or moved, the change in resource usage is automatically updated, and the new resource profiles are displayed. With this information available, the user can immediately see the effects of a schedule change on the mission resources. For each resource, PI2 also monitors any conflicts that are occurring on the resource.

Conflicts are time intervals where the limitations of the resource have been exceeded. These conflict intervals are highlighted in red to flag their existence for easy identification. Finally, PI2 monitors any dependencies that have been defined between activities and resources. The values of specific parameters of activities and resources may be functionally dependent on values of other parameters. PI2 automatically keeps these parameter values consistent.

PI2 also helps out by serving as an activity and resource database, producing/accepting information to/from a sequencer. The functional interface to PI2 has been extended to better assist an automated sequencer. A basic set of "fetch" functions have been developed to quickly retrieve information about conflicts and the resources and activities involved in the conflict. For example, an interface function has been written to fetch the legal times where an activity can occur in the schedule. Here, "legal times" refers to positions where no conflicts are caused by any of the resources used by the given activity.

In addition to fetching information about the current state of the schedule, the user will need to be able to change the current state in attempt to fix or optimize the schedule. Some basic primitive functions are provided by PI2 to allow an external system to add and move activities, change their duration, etc. These primitives make up the set of actions that a scheduler can take when trying to resolve conflicts.

Schedule Reasoner

The second major component of DCAPS is the automated schedule reasoner. This is the next step in automating and simplifying the spacecraft command sequencing process. There are three parts to the schedule reasoner: a schedule generator, a schedule repairer, and a schedule optimizer.

First, the schedule generator will transform a set of user-defined, high-level goals into a valid sequence of low-level commands. Second, the schedule repairer will automatically restore the consistency of the sequence after arbitrary user interaction by rescheduling using repair actions. The scheduler repairer iteratively attempts to resolve each conflict, which involves making choices on what to repair and how to repair it. Finally, the schedule optimizer can optimize a valid schedule to increase the scientific return.

Schedule Generator—The first step in sequencing spacecraft commands is to come up with an initial schedule of events for each phase of the mission. This process has been partially automated in DCAPS with the schedule generator. Expressing schedules and partial schedules to be generated is done through user defined goals. There are two ways in which user goals are handled in DCAPS. First, initial science and engineering goals are handled with parameterized scheduling functions. Each function implements a goal. For example, there is a "Place-Power" function that schedules power switching activities in appropriate places based on some engineering parameters. Parameters may include such things as a minimum time between switching, or a power on during a particular state of a different resource.

Second, science goals can also be expressed through data-take requests, which do not have to be a part of the initial schedule generation. For example, a scientist can request ten additional scans from a particular instrument to occur any time during some phase of the mission. This type of general request does not include specific locations or necessary supporting activities. The scheduler will simply place them at random positions and allow any conflicts to be resolved by the automated repairer.

Schedule Repairer—The generated initial schedule may still violate some of the spacecraft constraints. Also, the scientists and engineers might feel that their goals were not completely satisfied, and may need to interact with and modify the generated schedule. By modifying the schedule, new conflicts may be introduced. Therefore, we need some way of automatically resolving any existing conflicts in the schedule, while disrupting the current state of the schedule as little as possible. Having the process automated allows the user to be less careful, and therefore spend less time on the details of sequencing the activities. When general requests or changes have been made, all conflicts can be resolved by executing one simple command to invoke the schedule repairer.

Before describing the schedule repairer, we must present a few definitions. A "hard conflict," or just "conflict," is a violation of one of the resource constraints. A conflict occurs over a certain time period and is caused by activities called "culprits." For example, if the power capacity is exceeded from time t1 to time t2, then a conflict exists from time t1 to time t2, and the culprits are any activities that use power during this time (see Figure 3). A "soft conflict" is a violation of one of the user's high level goals. "Hard conflicts" are violations of legal constraints, while "soft conflicts" are violations of user preferences. "Choice points" are places in the scheduling algorithm when a decision must be made. For example, when there are many conflicts to resolve, the scheduler must decide which conflict to resolve first. A "hard choice," or just "choice," is a decision made solely on the basis of possible hard conflicts. It may be decided, for example, not to place an activity at a certain time because new conflicts will be added as a result of that placement. A "soft choice" is a decision made on the basis of user preferences or heuristics with the hopes of generating a more optimal schedule. An example of a user preference is a priority scheme on certain activities. One heuristic may be to move lowest-priority culprits to the nearest legal position.

There are three possible actions to take in attempt to resolve a conflict: move, add, or delete an activity. The "move" action involves moving one of the culprits of the conflict to a positions that will either resolve the conflict or at least ensure that the moved activity is no longer a culprit. Some conflicts can be resolved by adding a new activity. These activities usually provide some resource that was previously not available. Finally, a conflict can also be resolved by simply deleting the culprits. This is obviously not a preferred method and is only used as a last resort.

The resolution of a conflict greatly depends on the type of resource that is in violation. There are five different types of conflicts corresponding to the five types of resources. A state conflict occurs when an activity requires the resource to be in a state which it is not. The culprits in this type of conflict are all of the activities that require the incorrect state and the activity that changed the resource to the incorrect state. Several possibilities for resolving a state conflict include moving the culprits to another interval

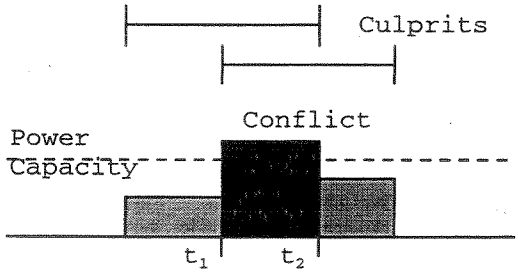


Figure 3: Conflicts

```
ResolveConflicts (max_iterations)
{
  iterations = 1
  conflicts = GetConflicts()
  Loop while (length(conflicts) > 0 &&
              iterations <= max_iterations) {
    conflict = ChooseConflict(conflicts)
    method = ChooseMethod(conflict)
    case (method) {
    'move'
      culprit = ChooseCulpritToMove(conflict)
      duration = ChooseDuration(conflict, culprit)
      start_time =
        ChooseStartTime(conflict,culprit,duration)
      success = MoveCulprit(conflict,culprit,start_time)
    'add'
      activity = ChooseActivityToAdd(conflict)
      duration = ChooseDuration(conflict, activity)
      start_time =
        ChooseStartTime(conflict,activity,duration)
      success = AddActivity(conflict,activity,start_time)
    'delete'
      culprit = ChooseCulpritToDelete(conflict)
      success = DeleteCulprit(conflict,culprit)
    }
    progress = GetProgress()
    if not(success || progress) then UndoLastAction()
    conflicts = GetConflicts()
    iterations = iterations + 1
  }
}
```

Figure 4: Iterative Repair Algorithm

where the required state is present or adding an activity that will change the state of the resource to the required state.

A concurrency conflict is when an activity requires the presence of the resource during a time for which it is absent. The culprits in this type of conflict are all of the activities that require the presence of the resource. To resolve a concurrency conflict, the scheduler can move the culprits to an interval where the resource is present or add an activity that provides the presence of the resource.

A depletable conflict means that the activities of the schedule have used too much of the resource. In this type of conflict, the culprit is the activity that caused the resource to overflow during the time that it first overflows. Some depletable resources have "resetter" activities and this sort of conflict can be resolved by adding an activity that "resets" the available resource. For example, a downlink activity will free up space in the downlink buffer. A nondepletable conflict is when activities overuse a resource during a particular time interval. The culprits in this type of conflict are all of the activities that use the resource during the conflict interval. This sort of conflict can be resolved by moving or deleting culprits. There are no activities in the DATA-CHASER model that can add to a nondepletable resource.

Simple conflicts occur when two or more activities use the same resource at the same time. This type of conflict can only be resolved by moving culprits.

For any type of initial schedule, the schedule repairer must find the correct activities to move, add, or delete and position them temporally in such a way that no conflicts remain. The scheduler makes decisions randomly except at certain choice points where heuristics are used. The scheduler relies on some interface functions to PI2 that describe the conflicts in the current schedule, describe the activities that could resolve a conflict, and manipulate the schedule. We first describe the random scheduler, followed by the heuristic enhancements that facilitate scheduling within the DATA-CHASER domain. The ultimate task of the system is to find the best place to schedule the activities so as to maximize the utility of the schedule. In the basic scheduler, all choices are made randomly from the list of options unless otherwise specified.

The algorithm is a simple iterative loop over the conflicts in the schedule (see Figure 4). First, a conflict is selected from the list of current conflicts. An attempt is made to resolve the chosen conflict. Next, a method for resolving the conflict is chosen. The repair action will depend on which method has been selected. If "move" is chosen, then a culprit must be picked from the list of culprits in the conflict. A duration and start time are chosen for the culprit, and the culprit is moved to the new location. If "add" is the chosen method, then the repairer must decide which activity type to instantiate. Again, a duration and start time must be chosen for the new activity, and the activity is inserted at the chosen time. If the repairer chooses to "delete" an activity, then it simply must choose an activity to delete, and delete it. After the chosen action is performed, the schedule repairer checks to see if progress was made. We define progress as either decreasing the number of conflicts, decreasing the number of culprits, or decreasing the duration of the conflicts.

If the action did not succeed in resolving the conflict, or progress was not made, then the action is "undone." Otherwise, the new set of conflicts are found, and the loop counter is incremented. This process continues until all conflicts are resolved, or the loop counter exceeds a user-defined maximum bound. For every choice point in the algorithm, where a selection must be made from a list of possibilities, the schedule repairer is allowed to backtrack to that point. What this means is, that if a particular choice fails, the schedule repairer may choose another from the list before giving up. If all choices fail, then a previous decision must have been incorrect, and the repairer can backtrack to the preceding choice point. All choice points, including the decision on whether or not to backtrack, are heuristic decisions and may customized to a particular domain.

Schedule Optimizer—The schedule optimizer is composed of additional knowledge supplied by the user and utilized by the other components of the scheduler. There are three ways to optimize a schedule: using preference heuristics at search choice points in the schedule repairer, specifying a set of "soft conflicts" for the repairer, and using an evaluation function to score results from multiple runs of the schedule generator and repairer.

A preference heuristic, or "soft choice," can be made at any decision in the repair search. For example, when deciding where to move a conflict causing activity, the user might prefer to move that activity to a position closest to its current position. This will help the scheduler avoid unnecessary disruption to the existing schedule. The existing schedule, after all, may have been produced by the user in an attempt to optimize the schedule.

Preferences can also be expressed using what we referred to as "soft conflicts." A soft conflict is a way of specifying a preferred value for a particular resource, possibly at a particular time. For example, having any scanned data that has not been stored on the tape at the end of the mission, is considered a soft conflict. This is not a hard conflict, because the data is not exceeding the buffer size. However, the scientist would prefer that all of the data be written to the tape at the mission's end, rather than leaving it in the on-board memory. After the schedule repairer handles all of the hard conflicts, it continues by iteratively addressing all of the soft conflicts.

The third approach to optimization involves scoring several resulting schedules and choosing the one with the highest score. The evaluation function is domain dependent and would have to be written separately for each application. Some basic scoring, however, will be similar across applications. For example, most science spacecraft are mainly concerned with collecting the largest number of images as possible. A simple evaluation would give a higher score to schedules with greater amounts of collected data. Once we have the evaluation function, we need to be able to produce several different schedules from the same goals and initial state.

This can be done by either changing the heuristics or by running the scheduler with a different random seed. Some heuristics may work better than others, and it is often difficult to tell which is the best for a particular application. Therefore, it may be necessary to resort to empirical tests. After running the scheduler on different heuristics, we can simply choose the set of heuristics which generates the schedule with the highest score. After choosing the heuristics, the scheduler can be run many times with different random seeds. At choice points where there is no heuristic for choosing from the list of possibilities, the scheduler makes a random decision. With different random seeds, these decisions will be different, and the resulting schedule will be different. Using the evaluation function, we can assign a score to each, and choose the schedule with the highest score. This procedure will not necessarily uncover the optimum schedule, but it will help find a more optimal schedule.

Heuristics—The general search and decision making described above would be futile without expert support and guidance. Heuristics have been developed and incorporated into DCAPS to help guide the search to a valid and more optimal schedule. This guidance knowledge comes from both domain experts and scheduling experts. There are three basic classes of heuristics used in DCAPS: selection, pruning, and backtracking heuristics.

Selection heuristics involve deterministically sorting or selecting from a list of possibilities at a choice point in the search. The selection is usually based on some property of the objects being considered. For example, when choosing a culprit to move in order to resolve a power conflict, one heuristic might choose the culprit that uses the most amount of power. Using this heuristic might resolve the conflict faster. Another successful heuristic used in DCAPS was one that sorted the possible locations for activity placement by the number of conflicts the activity would cause when placed in that location. This basic approach has been referred to as the "min-conflicts" heuristic [5]. The min-conflicts algorithm we use is interesting, and it is worthwhile to go into detail.

For each resource used by an activity, we query the database for the legal times where the activity can be placed without violating the resource constraint. Then, each legal interval is assigned an initial score of one. Next, we intersect two sets of intervals that resulted from two of the resources, using a special "scored" interval intersection (see Figure 5). The scored intersection of intervals A and B results in four possibilities: an interval with a score of A for positions where A exists and B does not, an interval with a score of B where B exists and A does not, an interval with a score of A plus the score of B where the two intervals intersect, or no interval where neither A nor B exist. The result of this intersection is then intersected with the third set of intervals.

This process continues until each set of intervals for each resource has been intersected. The result is a set of scored intervals, where the score represents the number of resources that will not be violated if the activity is placed in that position. Using these intervals, we can choose a

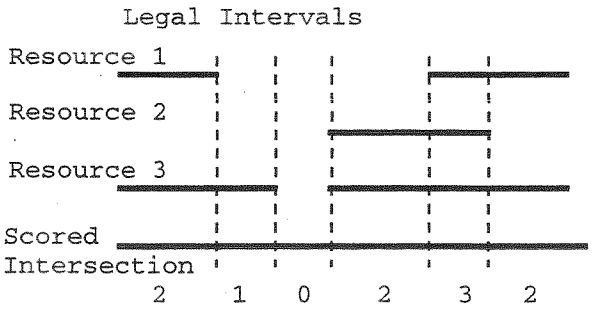


Figure 5: Min-conflicts with scored interval intersection

position with the highest score, in other words, the position with the fewest conflicts.

Another class of heuristics used in DCAPS are the pruning heuristics. These heuristics remove some of the possibilities for a given selection in attempt to make the choice easier and faster. For example, after finding the scored intervals for an activity, we may not want to try all possible positions. One possibility is to only try positions with the highest score or least number of conflicts. This process may speed up scheduling because the scheduler will only try a few positions before realizing this attempt is futile and giving up to try something different. Too much pruning, however, may remove possibilities that could be useful. In the above example, some of the pruned intervals may have included positions that, if the activity was placed there, would have improved the schedule. A more conservative approach might be to prune only those intervals that would cause more conflicts than are currently in the schedule. These intervals cannot possibly be positions that could improve the schedule.

Finally, backtracking heuristics are used to help determine when to continue working on the same problem and when to move on to a different problem. At each choice point, we have a list of possibilities. If we try one possibility, and it fails, we can continue and try the next possibility, or move on to a different choice point. Heuristics can be used to help make two types of decisions about backtracking: deciding on "action failures" and deciding on "selection failures." First, the notion of an "action failure" is not clear and requires an approximate definition. Success is not simply resolving the chosen conflict. When, resolving a conflict, and action attempt may fix the chosen conflict, but cause several other conflicts.

Therefore, success can be thought of as improving the schedule. But how much? And what defines an improvement? Our current definition of progress includes observing the change in the number of conflicts, the change in the number of culprits, and the change in the duration of the conflicts. Checking the progress of an action can be used as a heuristic for determining whether to accept the action, or try a different one. The second opportunity for heuristics comes when deciding if there is a "selection failure." While trying and failing on a list of possibilities for a choice point, at some point we must decide that the previous choice was a failure. Heuristics can help with this decision also.

System Integration

DCAPS will be integrated into the End-to-End Mission Operations System (EEMOS) that is currently being developed for the DATA-CHASER project as a prototype for the Pluto Express EEMOS [6]. Currently the DATA-CHASER EEMOS consists of seven parts: Command and Control, Fault/Event Detection Interaction Reaction (F/EDIR), DATA/IO (Data handling), the Ground Database, the Graphical User Interface, the software testbed, and finally the planning and scheduling system (DCAPS).

The command and control system that we are using, System Command Language (SCL, also known as Spacecraft Command Language), integrates procedural programming with a real-time, forward-chaining, rule-based system. DCAPS interfaces with SCL through DATA/IO by sending script scheduling commands that are to be scheduled either on the flight or ground system. This interface is implemented by mapping PI2 activities to SCL scripts that were written prior to flight and can be scheduled or event-triggered by activating rules. These scheduling and rule activation commands are then sent to DATA/IO which forwards that list to the SCL Compiler. Once compiled, the list is sent to the payload through the next available uplink.

DCAPS is also interfaced with the ground EEMOS database, O2. O2 is an object-oriented database that will be used to store all mission data and telemetry that is downlinked by the payload. It will also store a command history. Through DATA/IO, DCAPS will request current payload status data in the form of sensor values in the telemetry history. It will also request lists of all commands uplinked during a given time interval. These are used by DCAPS to infer command completion status as well as to get the current state of the payload so that a new schedule can be created.

During mission operations, approximately every four hours or so, DCAPS will be asked by an operator to generate script scheduling commands and rule activations for the next six hours according to its schedule. Once this list is finished, it is reviewed by the Mission Operations staff on duty. If judged to be correct, scheduling and rule activation commands will be sent to DATA/IO during the next available uplink window.

If during that six hour period there is a major change in the NASA activities, DCAPS will ask if the users want to update the schedule script on-board. If the user accepts it, DCAPS will generate a updated list, ask the user to verify it, and send the list to DATA/IO to be uplinked.

Summary and Related Work

Iterative algorithms have been applied to a wide range of computer science problems such as traveling salesman [7] as well as Artificial Intelligence Planning [8,9,10,11]. Iterative repair algorithms have also been used for a number of scheduling systems. The GERRY/GPSS system [1,12] uses iterative repair with a global evaluation function and simulated annealing to schedule space shuttle ground processing activities. The Operations Mission Planner (OMP) [13] system used iterative repair in combination with a historical model of the scheduler actions (called chronologies) to avoid cycling and getting caught in local minima. Work by Johnston and Minton [5] shows how the min-conflicts heuristic can be used not only for scheduling but for a wide range of constraint satisfaction problems. The OPIS system [14] can also be

performing iterative repair. However, OPIS is more informed in the application of its repair methods in that it applies a set of analysis measures to classify the bottleneck before selecting a repair method.

In summary, DCAPS represents a significant advance from several perspectives. First, from a mission operations perspective, DCAPS is important in that it significantly reduces the amount of effort and knowledge required to generate command sequences to achieve mission operations goals. Second, from the standpoint of Artificial Intelligence applications, DCAPS represents a significant application of planning and scheduling technology to the complex, real-world problem of spacecraft commanding. Third, from the standpoint of Artificial Intelligence Research, DCAPS mixed initiative approach to initial schedule generation, iterative repair, and schedule optimization represents a novel approach to solving complex planning and scheduling problems.

Status Note

This paper was written based on the status of the DATA-CHASER project as of April 1997. There have been recent changes to the system architecture due to last minute problems integrating the software with the flight hardware. Some components, such as the SXEE and LASIT science instruments, will not be operational, and therefore not scheduled, during nominal operations. However, post-flight simulations and testing will be done on the complete, original system architecture.

Acknowledgments

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

References

[1] M. Zweben, B. Daun, E. Davis, and M. Deale, "Scheduling and Rescheduling with Iterative Repair," in Intelligent Scheduling, Morgan Kaufman, San Francisco, 1994.

[2] DATA-CHASER Documents, Annual Report.

[3] G. Rabideau, S. Chien, T. Mann, C. Eggemeyer, P. Stone, and J. Willis, "DCAPS User's Manual," JPL Technical Document D-13741, 1996.

[4] W. Eggemeyer, "Plan-IT-II Bible", JPL Technical Document, 1995.

[5] M. Johnston and S. Minton, "Analyzing a Heuristic Strategy for Constraint Satisfaction and Scheduling," in Intelligent Scheduling, Morgan Kaufman, San Francisco, 1994.

[6] S. Siewert and E. Hansen, "A Distributed Operations Automation Testbed to Evaluate System Support for Autonomy and Operator Interaction Protocols," 4th International Symposium on Space Mission Operations and Ground Data Systems, ESA, Forum der Technik, Munich, Germany, September, 1996.

[7] S. Lin and B. Kernighan, "An Effective Heuristic for the Traveling Salesman Problem," Operations Research Vol. 21, 1973.

[8] S. Chien and G. DeJong, "Constructing Simplified Plans via Truth Criteria Approximation," Proceedings of the Second International Conference on Artificial Intelligence Planning Systems, Chicago, IL, June 1994, pp. 19-24.

[9] K. Hammond, "Case-based Planning: Viewing Planning as a Memory Task," Academic Press, San Diego, 1989.

[10] R. Simmons, "Combining Associational and Causal Reasoning to Solve Interpretation and Planning Problems," Technical Report, MIT Artificial Intelligence Laboratory, 1988.

[11] G. Sussman, "A Computational Model of Skill Acquisition," Technical Report, MIT Artificial Intelligence Laboratory, 1973.

[12] M. Deale, M. Yvanovich, D. Schnitzius, D. Kautz, M. Carpenter, M. Zweben, G. Davis, and B. Daun, "The Space Shuttle Ground Processing System," in Intelligent Scheduling, Morgan Kaufman, San Francisco, 1994.

[13] E. Biefeld and L. Cooper, "Bottleneck Identification Using Process Chronologies," Proceedings of the 1991 International Joint Conference on Artificial Intelligence, Sydney, Australia, 1991.

[14] S. Smith, "OPIS: A Methodology and Architecture for Reactive Scheduling," in Intelligent Scheduling, Morgan Kaufman, San Francisco, 1994.

ASPEN: A Framework for Automated Planning and Scheduling of Spacecraft Operations

Gregg Rabideau and Alex S. Fukunaga and Steve Chien and David Yan

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, MS 525-3660
Pasadena, CA 91109-8099
{alex.fukunaga,gregg.rabideau,steve.chien,david.yan}@jpl.nasa.gov

Abstract

A number of successful applications of automated planning and scheduling of spacecraft operations have recently been reported in the literature. However, these applications have been one-of-a-kind applications that required a substantial amount of development effort. In this paper, we describe ASPEN (Automated Planning/Scheduling Environment), a modular, reconfigurable application framework which is capable of supporting a wide variety of planning and scheduling applications. We describe the architecture of ASPEN, as well as a number of current spacecraft control/operations applications in progress.

Introduction

Automated planning/scheduling technologies show great promise in reducing operations cost and increasing the autonomy of aerospace systems. Planning[1] is the selection and ordering of activities such that they achieve one or more goals and satisfy a set of domain constraints. Scheduling selects among alternative plans and assigns resources and times for each activity so that the assignments obey the temporal restrictions between activities and the capacity limitations of a set of shared resources. In addition, scheduling is an optimization task in which metrics such as tardiness and makespan (overall schedule execution time) are minimized. Scheduling is a classical combinatorial problem that has long been studied by researchers in operations research. While traditional operations research approaches (c.f. (Graves 1981)) have focused on optimal solutions for highly restricted classes of problems, there has been much recent interest in the heuristic, constraint-based approaches that are applicable to practical domains.

Traditionally, the problems of planning and scheduling have been studied separately. Recently, approaches that integrate both the planning and scheduling process together under a unifying framework in which plans are generated and scheduled simultaneously by a single system (as opposed to using separate planning and scheduling systems) have been developed. Some recent aerospace applications of hybrid planner/schedulers include (Muscettola 1994; Muscettola *et al.* 1997; Rabideau *et al.* 1997).

Although the benefits of applying planning/scheduling technology can be significant, developing real-world, planning/scheduling systems is often an extremely time-consuming task. Modeling a complex domain requires an expressive modeling language, as well as data structures that represent the constraints expressed in the domain model. In addition, complex data structures and algorithms that support incremental modifications to candidate plans/schedules need to be designed and implemented.

In order to enable the rapid development of automated scheduling systems for NASA applications, we have developed ASPEN (Automated Scheduling and Planning ENvironment), a reusable, configurable, generic planning/scheduling application framework. An application framework (Pree 1995) is a class library (i.e., a reusable set of software components) that provides the functionality of the components found in prototypical instances of a particular application domain. Frameworks anticipate much of an application's design, which is reused in all applications based on the framework. In order to facilitate code reuse, ASPEN (as with many frameworks) incorporates multiple "design patterns" (Gamma *et al.* 1995). This implies a significant reduction in the amount of code necessary to implement successive systems.

The reusable components provided by ASPEN include:

- An expressive constraint modeling language to allow the user to naturally define the application domain;
- A constraint management system for represent-

[1] We take these definitions of planning/scheduling from (Fox 1994).

ing and maintaining operability and resource constraints, as well as activity requirements;

- A temporal reasoning system for expressing and maintaining temporal constraints;
- A set of search engines for constructing, repairing, and optimizing plans/schedules;
- Linear Programming utilities for optimizing schedule preferences; and
- A graphical interface for visualizing plans/schedules (for use in mixed-initiative systems in which the problem solving process is interactive).

ASPEN is currently being utilized in the development of an automated planner/scheduler for commanding the New Millennium EO-1 satellite and a naval communications satellite, as well as a scheduler for the ground maintenance for the Reusable Launch Vehicle and a design analysis tool for the Pluto Express spacecraft. The rest of the paper is organized as follows: First, we describe the architecture of ASPEN and its components. Next, we describe some current applications of the ASPEN framework, including ground maintenance scheduling for the Reusable Launch Vehicle, as well as operations planning/scheduling for two autonomous satellites. Finally, we describe related work.

The ASPEN Architecture

The development of an application framework for a particular domain implies a standardized approach to implementing systems for that domain, and a commitment by the framework developer to support applications that conform to that standardized approach. It is impractical to develop a framework to support all viable approaches to planning and scheduling. Numerous, widely divergent approaches to planning and scheduling have been developed (cf. (Allen, Hendler, & Tate 1990; Zweben & Fox 1994). Since planning/scheduling are currently very active areas of research, there is no clear consensus on which approaches are most useful. Thus, we restricted the scope of our framework to approaches that had been found useful for NASA applications in the past.[2]

By analyzing our previous experience with building planning/scheduling systems, (cf. (Rabideau *et al.* 1997; Muscettola *et al.* 1997)), as well as requirements for current and future applications, we abstracted a set of requirements that is flexible enough to support a wide range of applications, and developed the components shown in Figure 1 and described below:

[2]See (Chien *et al.* 1997) for an overview of planning/scheduling applications recently developed at JPL.

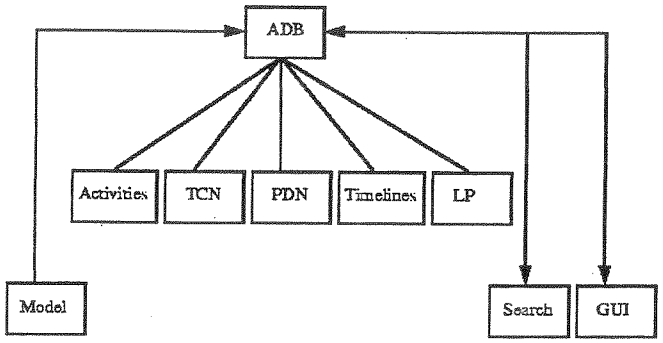


Figure 1: The ASPEN Architecture. The model feeds in the problem specification. The GUIs and search engines allow manual or automatic manipulation of the schedule, respectively. The ADB interfaces to a set of typical planning/scheduling toolkits.

Activity Database

The central data structure in ASPEN is an activity. An activity represents an action or step in a plan/schedule. An activity has a start time, end time, and a duration. Activities can use one or more resources. All activities in a plan/schedule are elements of the Activity Database (ADB), which maintains the state of all of the activities in the current plan/schedule, and serves as the integrating component that provides an interface to all of the other classes.

One function of the ADB is to represent and maintain hierarchical relationships between activities. Activities can contain other activities as subactivities; this facility can be used to reason about the plan/schedule at various levels of abstractions (i.e., a scheduler can first reason about a set of activities without considering that each of those activities are themselves composed of a set of subactivities). This can make various reasoning tasks much more computationally tractable.

Temporal and resource constraints between activities are also represented in the ADB. Although most of the actual computational mechanisms that maintain these constraints are implemented in the other modules described below, the protocol that a search algorithm uses to access constraints in the context of a plan/schedule is implemented in the ADB. For example, although the resource timelines are responsible for detecting overuse of resources by activities, the ADB maintains data structures that indicate the assignment of activities to specific timelines, so that one can efficiently ask queries such as, "which resources does this activity use?"

As another example: although the temporal con-

straint network (see below) is responsible for maintaining temporal constraints between individual activities, the ADB is responsible for global constraints. (e.g., the ADB contains global constraints such as: "all activities occur after the start of the scheduling horizon." When an activity is created, the temporal constraint that the activity occurs after the horizon is created automatically by the ADB).

Temporal Constraint Network

A Temporal Constraint Network (TCN) is a graph data structure that represents temporal constraints between activities. A temporal constraint describes the temporal relationship between an activity and other activities and/or the scheduling horizon, and imposes an ordering on the set of activities. The TCN implements a Simple Temporal Problem, as defined in (Dechter, Meiri, & Pearl 1991), and represents a set of constraints, all of which must be satisfied at any given time, i.e., it represents the conjunct of all active constraints between activities in the ADB. Activities can represented in the TCN as pairs of time points, where each time point corresponds to the beginning or end of an activity, and the edges in the TCN graph represent the constraints on the temporal distance between the time points. The TCN can be queried as to whether the temporal constraints currently imposed between the activities are consistent.

Resource Timelines

Resource timelines are used to reason about the usage of physical resources by activities. Capacity conflicts are detected if the aggregate usage of a resource exceeds its capacity at any given time. Several subclasses of resource timelines are implemented, including depletable resource timelines used to model consumable resources (e.g., fuel), and non-depletable resources that are used to model resources which are not actually consumed by usage, but are instead "reserved" for a period of time (e.g., a piece of equipment). Our current model of resource usage is discrete. That is, if we specify that an activity such as move-forward uses 2 units of fuel, then both of these units are modeled as being immediately consumed at the beginning of the activity. This is a discrete approximation, since the usage of the fuel may be better modeled as a linear function such as $usage(t) = 2t/(activitylength)$, where t is the time elapsed since the beginning of the activity, and $activitylength$ is the duration of the activity.

State Timelines

State timelines represent arbitrary attributes, or states, that can change over time. Each state can have several possible values; at any given time, a state has exactly one of these values. Activities can either change or use states. For example, a door-open activity would set the state of door to be open, while an enter-building activity would require that the state of door be open. As activities are placed/moved in time, the state timeline updates the values of the state, and detects possible inconsistencies or conflicts that can be introduced as a result. For example, an activity that requires that the door be open is placed at time t, then the state timeline checks to verify that the door is in fact open at time t. Otherwise, a state constraint violation is indicated. Users can define legal sequences of state transitions. The state timeline class will detect illegal transition sequences if they are introduced into the timeline.

For example, consider modeling a traffic light with a state timeline, *traffic-light*. The possible values are *green*, *yellow*, and *red*. The legal state value transitions are: *green* to *yellow*, *yellow* to *red*, *red* to *green*. All other transitions are illegal.

Parameter Dependency Network

Each activity has a number of parameters that are either user-defined or computed by the system, such as start time, end time, duration, any resources it uses, any states it changes/uses, etc. In ASPEN, it is possible to create dependencies between pairs of parameters within the same activity, or between pairs of parameters defined in different activities. A dependency between two parameters p_1 and p_2 is defined as a function from one parameter to another, $p_1 = f(p_2)$, where $f(x)$ is an arbitrary function whose input is the same type as p2, and whose output has the type of p1. These dependencies can be represented and maintained in a Parameter Dependency Network (PDN). The PDN maintains all dependencies between parameters, so that at any given time, all dependency relations are satisfied. Note that if there exists a dependency such that $p_1 = f(p2)$, its inverse dependency, $p_2 = f^{-1}(p_1)$ does not necessary exist, unless the user specifies the inverse relationship and specifies the inverse dependency as well.

Note that the TCN can be seen as a special case of a PDN in which the functional relationships between the parameters (activity start/end times and durations) is a distance relationship, and for which very efficient constraint propagation algorithms have been implemented.

In general, as commonly used special cases of functional dependencies between parameters (such as temporal distance relationships), it can be useful to develop special dependency networks that implement efficient constraint propagation algorithms that take advantage of the special structure of these dependencies,

instead of relying on the general mechanism offered by the PDN.

Such special-purpose dependency networks can be implemented as subclasses of the abstract parameter dependency network, or (if the protocol that must be supported is sufficiently unique) abstracted out as a separate basic component of ASPEN, as was done with the TCN.

Planning/Scheduling Algorithms

The search algorithm in a planning/scheduling system searches for a valid, possibly near-optimal plan/schedule. The ASPEN framework has the flexibility to support a wide range of scheduling algorithms, including the two major classes of AI scheduling algorithms: constructive and repair-based algorithms. We have implemented one of each of these types of algorithms.

Constructive algorithms (e.g., (Fox 1994)) incrementally construct a valid schedule, ensuring that at every step, the partial schedule constructed so far is valid. When a complete schedule is constructed, it is therefore guaranteed to be valid. Repair-based algorithms (cf. (Minton *et al.* 1988; Zweben *et al.* 1994)) start with a possibly invalid complete schedule, and at every iteration the schedule is analyzed and repair operations that attempt to eliminate conflicts in the schedule are applied until a valid schedule is found.

The constructive algorithm we have implemented for ASPEN uses a forward sweeping dispatch procedure. The first step is to plan for the requested activities, or goals. This may include creating new supporting activities and making the links between related activities, but without assigning specific start times. Then, all activities are sorted in increasing order by their latest possible start time. In this order, each activity is assigned a fixed start time and placed on the timelines, making the necessary reservations.

We have also developed an iterative repair algorithm for resolving all types of conflicts in ASPEN. It is similar to the DCAPS iterative repair (Rabideau *et al.* 1997), but with extensions for handling new types of conflicts. Conflicts arise when a constraint is violated. A temporal constraint is violated when it is either not satisfied, or incorrectly satisfied. A resource is violated when its capacity bounds are exceeded. A state variable is violated when an activity requires an inconsistent state or makes an illegal transition. A parameter constraint is violated when a parameter value is not consistent with the function used to compute the parameter. The operational constraint is violated when an activity does not directly correspond to an executable command.

At each iteration, the repairer must make decisions for a series of choice points. It must decide which conflict to address and what operation to perform to resolve the conflict. The possible operations include primitives such as moving, adding, and deleting activities. In addition, multiple step operations have been defined, which consist of a sequence of primitives. Depending on which operation was chosen, there may be more choices for the particular parameters of the operation. For example, if we choose to move an activity to resolve the conflict, we must decide which activity to move and where to move it. For each choice point, there is a function for fetching the entire list of possible choices. Each choice point may also have a corresponding heuristic function, which sorts and possibly prunes the given list of all choices. After making the necessary decisions, the repair operation is performed in an attempt to resolve the conflict. At this point, if conflicts exist and we have not exceeded the maximum number of iterations, then the new set of conflicts is retrieved and the process is repeated.

We are currently working on optimization algorithms for ASPEN. We have implemented a basic utility assignment scheme, which allows users to specify preferences and assign utilities to model objects (e.g. activities) in ASPEN. With this capability in place, we are developing an algorithm that will iteratively attempt to maximize the utility of the schedule. This would provide "any-time" access to a consistent schedule, where utility increases with the number of iterations (i.e. the longer the algorithm runs). We are also developing methods to incorporate some well-defined, stable linear programming algorithms to globally optimize user preferences.

Graphical User Interface

The ASPEN Graphical User Interface (GUI) component (Figure 2) provides tools for graphically displaying and manipulating schedules. Resource and state timelines are displayed. Activities are overlayed on the timelines, and users can directly manipulate activities using standard drag-and-drop procedures.

Extending ASPEN for Applications

There are two means by which ASPEN can be extended and specialized for a particular application. These are:

- Creation of domain-specific models using the modeling language, and
- Extension of the application framework code.

The modeling language is used to specify domain-specific constraints and activities. Figure 3 shows part

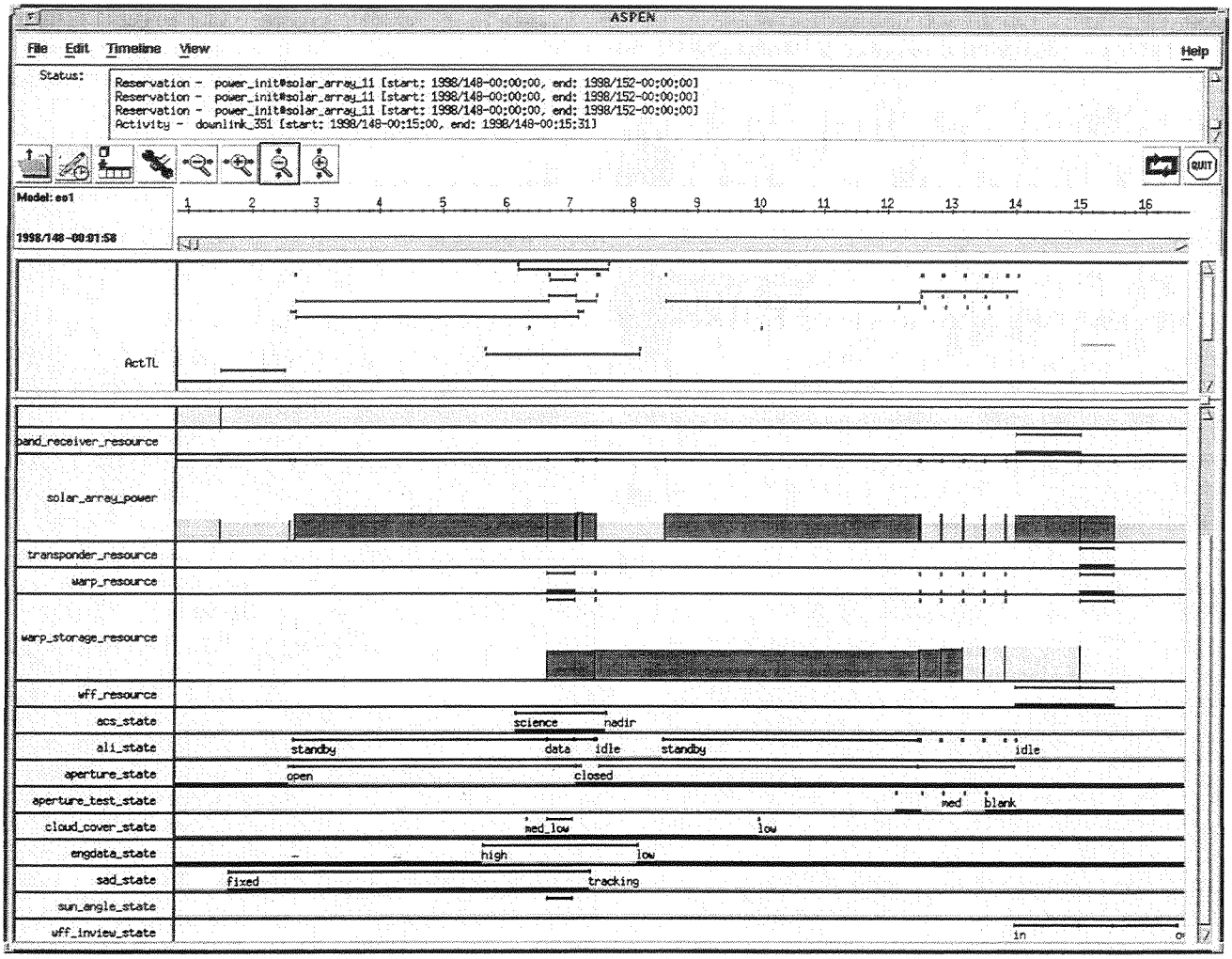


Figure 2: ASPEN GUI. Top section has activities as horizontal bars. Middle section has resource timelines. Bottom section has state variable timelines.

of a domain model specified in the ASPEN modeling language.

The base ASPEN framework, including the modeling language is sufficiently extensible to support a range of applications without any extensions to the code of the framework itself (e.g., the Reusable Launch Vehicle ground maintenance scheduling application is directly derived from the framework by simply specifying a model file).

Extensions to the framework code need to be made when changes in the behavior of ASPEN components are required. This includes two classes of extensions: epistemological and heuristic.[3] Epistemological extensions are necessary when new representational capabilities are in order to model a new domain. For example, if we wanted to implement a new type of resource timeline which had a more sophisticated, continuous model of resource usage[4], then a new subclass of the resource timeline abstract class would need to be implemented. Heuristic extensions customize the behavior of the framework to improve the quality of solutions found or the time to find good solutions for a particular domain.[5] Examples of heuristic extensions include new repair heuristics for a repair-based scheduler, or an entirely new search algorithm.

[3]This classification follows (McCarthy & Hayes 1969).

[4]Recall from Section 3 that we use a simple, discrete resource usage model.

[5]This implies that the framework is, in principle capable of eventually finding some solution without a heuristic extensions.

```
Activity prevalve_removal {
  duration = [15, 20]
  slot subsystem ss
  after prevalve_prep with (ss = subsystem)
  before prevalve_replace with (ss = subsystem)
  reservation hydraulic_lift use 1
  reservation prevalve must_be purged
  reservation prevalve_area must_be illuminated
}

Resource hydraulic_lift {
  type non-depletable
  capacity 1
}

State_Variable prevalve {
  states purged unpurged
  transitions purged to_and_back unpurged
}
```

Figure 3: Sample of ASPEN modeling language (part of the Reusable Launch Vehicle maintenance model). This describes an activity for removing the prevalve of an engine subsystem.

Applications of ASPEN

In this section, we describe ongoing applications of the ASPEN scheduling system to: generation of spacecraft command sequences for the New Millennium Earth Orbiter One satellite and the U.S. Navy UHF Follow On One satellite; generation of mission operations sequences to assist in design analysis for science and operability; and rapid generation of plans for maintenance and refurbishing for Highly Reusable Space Transportation.

Spacecraft Commanding

The primary application area for the ASPEN scheduling system is generation of spacecraft command sequences from high level goal specifications.

In this role, automated scheduling systems will enable encoding of complex spacecraft operability constraints, flight rules, spacecraft hardware models, science experiment goals and operations procedures to allow for automated generation of low level spacecraft sequences by use of planning and scheduling technology.

By automating this process and encapsulating the operation specific knowledge we hope to allow spacecraft commanding by non-operations personnel, hence allowing significant reductions in mission operations workforce with the eventual goal of allowing direct user commanding (e.g., commanding by scientists).

Current ASPEN applications to spacecraft commanding focus on two missions: the New Millennium Earth Orbiter One (EO-1) satellite (Sherwood *et al.* 1997b) (to be launched in late 1998) and the U.S. Navy UHF Follow On One (UFO-1) satellite (currently in orbit). EO-1 (Speer *et al.* 1997) is a earth imaging satellite featuring an advanced multi-spectral imaging device. For this mission, operations consists of managing spacecraft operability constraints (power, thermal, pointing, buffers, consumables, engineering downlinks, etc.) and science goals (imaging of specific targets within particular observation parameters). Of particular difficulty is managing the downlinks as the amount of data generated by the imaging device is quite large and ground contacts are a limited resource.

The current ASPEN EO-1 scheduler generates an initial schedule using forward sweeping greedy dispatch to generate an initial schedule, then uses the extension of the DCAPS iterative repair algorithm to resolve state, resource, and temporal conflicts.

Another ongoing effort in the area of spacecraft commanding is the development of an advanced commanding system for the U.S. Navy UFO-1 satellite (UFO-1 1997). UFO-1 is an on-orbit testbed managed by the U.S. Naval Academy Space Artificial Intelligence Lab (SAIL) at Annapolis. In this collaboration, SAIL is developing an uplink, downlink, basic data transport, and commanding capability to be interfaced with an advanced planning and scheduling engine (ASPEN). In this application, ASPEN will allow high level commanding of the UFO-1 satellite to perform high level functions such as: auto pitch momentum dumping, preparation for eclipse season, delta-V maneuvers, IRU warmup and turnon, battery cell pressure bias calibration, delta inclination maneuvers, and other engineering housekeeping functions. The ASPEN scheduling engine then performs appropriate expansion and conflict resolution to generate lower level command sequences to achieve the higher level goals.

Design Evaluation

ASPEN is also being applied in the Pluto Express (PX) for the dual purposes of science planning and design evaluation for science and operability (PX 1996; Sherwood *et al.* 1997a). In support of science planning, we are developing high level models of proposed PX spacecraft to assist in automated generation of science data acquisition plans (e.g., high level activity sequences) from high level science goals to assist in developing science plans for mission profiling.

This same capability to generate science plans is being used to evaluate candidate spacecraft designs from the standpoint of emergent design aspects such as science return and operability. This spin-off application arises from the observation that often it is difficult

to determine how well a given spacecraft design will perform without fleshing out approximate operations sequences for critical phases of the mission (e.g., encounter). In order to address this difficulty, we are developing a design analysis tool which accepts as input: a candidate spacecraft design (and operations constraints, models, etc.); a set of engineering and science objectives; and a set of scoring functions to assess how well a sequence achieves the objectives. This tool then applies an ASPEN-based planner/scheduler to generate a candidate sequence; then uses the scoring function to score this sequence in terms of the aspects of science, operability, etc. This enables design teams to rapidly and impartially evaluate large numbers of spacecraft designs with little effort, thus allowing improved analysis of design tradeoffs to enhance science and operations concerns for future missions.

Maintenance Scheduling

As part of the NASA Highly Reusable Space Transportation (HRST) program[6] we have been developing and demonstrating advanced scheduling systems for the rapid generation and revision of plans for maintenance and refurbishment of highly reusable launch vehicles (Fukunaga *et al.* 1997). In this application, real-time telemetry downlinked either during flight or immediately after flight would be analyzed to automatically generate a set of maintenance requests, which would then be transformed into a refurbishment plan by an automated planning and scheduling system which would account for available equipment and resources as well as the intricacies of the refurbishment procedures of the highly complex propulsion systems. The end target is to allow a turnaround of several hours for the HRST spacecraft to support a flight frequency on the order of several flights per day.[7] If the maintenance schedule can be generated using in-flight telemetry then the refurbishment process can be sped up even further by allowing for downlinking of requests for pre-positioning of equipment and resources to minimize schedule delay.

Once the actual maintenance plan has been generated, the planning tool continues to be of use in two ways. First, in many cases there can be several mutually exclusive maintenance activities which can be performed. Via lookahead and critical path analysis automated scheduling software can determine the next activities to enable the minimal makespan. Also, as unexpected events arise (such as equipment failures, resource unavailabilities, and schedule slippage), the automated scheduling software has the ability to revise the schedule so as to minimize schedule disruption (movement of activities and resources from their original assignments) and schedule slippage (delay of the completion of the overall refurbishment).

In order to test and validate this technology we have been utilizing test maintenance procedures. Specifically, we used the maintenance procedures developed for the LO_2 and LH_2 propulsion systems for the Rockwell International X-33 Reusable Launch Vehicle.[8] The procedures derived for maintaining and refurbishing the test articles provided a rich testbed for Space Propulsion System Maintenance Scheduling. Our testbed model consisted of 576 activity types, 6 resources, and on average 6 state, resource, and precedence constraints per activity. In this application we allowed maintenance requests to request either refurbishment of specific subsystems or major systems. In order to schedule the maintenance requests the ASPEN system used a forward sweeping greedy dispatch algorithm which used strong knowledge of the precedences of activities in the plan. The resulting scheduler has been able to generate schedules for refurbishment problems involving approximately half of the subsystems (8 subsystems, 358 activities) in 8 minutes.

Related Work

The idea of an application framework for planning/scheduling was pioneered in the OZONE system of Smith et al. (Smith & Lassila 1994; Smith, Lassila, & Becker 1996), which has been used in production management, transportation scheduling, and logistics applications. Differences between OZONE and ASPEN include the following:

- OZONE has emphasized applications in manufacturing and transportation planning and scheduling, while ASPEN is designed for spacecraft operations domains.
- OZONE emphasizes decision support tools, while ASPEN (due to the nature of spacecraft operations domains) emphasizes tools that support more autonomous decision-making applications.

Conclusions and Future Work

In this paper, we have described ASPEN, a reconfigurable, modular framework for planning/scheduling

[6] Which targets the development of technologies enabling highly reusable, low-cost space transportation systems (NAS 1995; 1996).

[7] In comparison, the space shuttle refurbishment process takes approximately 65 days with a flight frequency of once per 4 months; the current Reusable Launch Vehicle initiative has a targeted flight frequency of once every 1-2 weeks.

[8] Developed by Rockwell International during the Phase 1 competition of the Reusable Launch Vehicle Program which ended in July 1996.

applications, and described three current applications of ASPEN in spacecraft operations. Although the development of a generic software architecture has required a substantial, initial investment of effort, we expect the total development effort for a set of scheduling applications to be significantly less as compared to individually developing each of the applications.

ASPEN is currently a scheduling-oriented system, although some planning capabilities are supported for hybrid planning/scheduling applications such as the EO-1 and UFO-1. ASPEN already supports much of the functionality of classical planning systems (Allen, Hendler, & Tate 1990). We plan to extend ASPEN's planning capabilities so that it can be used as a framework for planning applications, while maintaining the additional temporal reasoning and resource management capabilities which are available through ASPEN's scheduling-oriented facilities.

Acknowledgments

This work was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract to the National Aeronautics and Space Administration. We thank Stephen F. Smith for helpful comments on a draft of this paper. Robert Sherwood implemented spacecraft operations models for the ASPEN applications, and Quoc Vu implemented the GUI.

References

Allen, J.; Hendler, J.; and Tate, A. 1990. *Readings in Planning.* Morgan Kaufmann.

Chien, S.; DeCoste, D.; Doyle, R.; and Stolorz, P. 1997. Making an impact: Artificial intelligence at the Jet Propulsion Laboratory. *AI Magazine* 18(1):103–122.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Fox, M. 1994. Isis: a retrospective. In Zweben, M., and Fox, M., eds., *Intelligent Scheduling.* Morgan Kaufmann.

Fukunaga, A.; Rabideau, G.; Mann, T.; Winkler, D.; Chien, S.; and DeCoste, D. 1997. Automated telemetry analysis, maintenance request generation, and maintenance plan generation, scheduling, and updating for highly reusable space transportation: A preliminary demonstration. In *Proceedings of the NASA Workshop on Planning and Scheduling.* Oxnard, CA: NASA.

Gamma, E.; Helm, R.; Johnson, R.; and Vlissides, J. 1995. *Design patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley.

Graves, S. 1981. A review of production scheduling. *Operations Research* 29(4):646–675.

McCarthy, J., and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., and Mitchie, D., eds., *Machine Intelligence*, volume 4. Edinburgh University Press. 463–502.

Minton, S.; Johnston., M.; Philips, A.; and Laird, P. 1988. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence* 58:161–205.

Muscettola, N.; Smith, B.; Chien, S.; Fry, C.; Rajan, K.; Mohan, S.; Rabideau, G.; and Yan, D. 1997. On-board planning for new millennium deep space one autonomy. In *Proceedings of IEEE Aerospace Conference*, 303–318.

Muscettola, N. 1994. Hsts: Integrating planning and scheduling. In Zweben, M., and Fox, M., eds., *Intelligent Scheduling.* Morgan Kaufmann.

NASA. 1995. *Proceedings of the Technical Interchange Meeting on Highly Reusable Space Transportation*, Huntsville, AL.

NASA. 1996. *Proceedings of the Technical Interchange Meeting on Highly Reusable Space Transportation*, Huntsville, AL.

Pree, W. 1995. *Design Patterns for Object-Oriented Software Development.* ACM Press.

PX. 1996. Pluto express information page, *http://www.jpl.nasa.gov/pluto/.*

Rabideau, G.; Chien, S.; Mann, T.; Willis, J.; Siewert, S.; and Stone, P. 1997. Interactive, repair-based planning and scheduling for shuttle payload operations. In *Proceedings of IEEE Aerospace Conference*, 325–341.

Sherwood, R.; Chien, S.; Rabideau, G.; and Mann, T. 1997a. Design for x (dfx): Operations characteristics spacecraft design analysis tool. In *Proceedings of the NASA Workshop on Planning and Scheduling.* Oxnard, CA: NASA.

Sherwood, R.; Govindjee, A.; Yan, D.; Rabideau, G.; Chien, S.; and Fukunaga, A. 1997b. Aspen: Eo-1 mission activity planning made easy. In *Proceedings of the NASA Workshop on Planning and Scheduling.* Oxnard, CA: NASA.

Smith, S., and Lassila, O. 1994. Configurable systems for reactive production management. In Szelke, E., and Kerr, R., eds., *Knowledge-Based Reactive Scheduling, International Federation of Information Processing (IFIP) Transactions B-15.*

Smith, S.; Lassila, O.; and Becker, M. 1996. Configurable mixed-initiative systems for planning and scheduling. In Tate, A., ed., *Advanced Planning Technology.* AAAI Press.

Speer, D.; Hestness, P.; Perry, M.; and Stabnow, B. 1997. The new millennium program eo-1 mission and spacecraft design concept. In *Proceedings of IEEE Aerospace Conference*, volume 4, 207–227.

UFO-1. 1997. Ultra high frequency follow on communications satellite system fact sheet, *http://www.fafb.af.mil/fact_sheets/index.html.*

Zweben, M., and Fox, M. 1994. *Intelligent Scheduling.* Morgan Kaufmann.

Zweben, M.; Daun, B.; Davis, E.; and Deale, M. 1994. Scheduling and rescheduling with iterative repair. In Zweben, M., and Fox, M., eds., *Intelligent Scheduling.* Morgan Kaufmann.

The Overall Architecture of the HST Science Planning and Scheduling System

Roberto J. Samson

Space Telescope Science Institute
3700 San Martin Drive
Baltimore, MD 21218
samson@stsci.edu

Abstract

This paper presents an overview of the architecture and function of the Hubble Space Telescope (HST) Science Planning and Scheduling System (SPSS). SPSS is the system responsible for scheduling HST activities that must be in compliance with scientific as well as spacecraft health and safety constraints. The paper will focus on describing the constraint data and processing involved in producing flight calendars.

Introduction

The Science Planning and Scheduling System (SPSS) performs the detailed and accurate scheduling of Hubble Space Telescope (HST) activities that are required to capture the science data specified in groups of proposals. The activities must be scheduled in compliance with constraints imposed for spacecraft health and safety and science integrity, as well as maximize the science data returned by the telescope. Generally, activities are scheduled to the nearest second with pointing uncertainties of well under one arcsecond. These activities are normally scheduled a few weeks in advance of actual flight.

HST activities are nominally scheduled over a seven-day period and are defined in an SPSS "**calendar**" file. The flight calendar builder creates a calendar from a list of "**candidate**" observations selected from a pool maintained in a relational database. Prior to the calendar building stage, proposal instructions are transformed to internal data structures and undergo rigorous validation and planning phases.

This paper provides an overview of the process involved in producing HST flight calendars. It will focus on the SPSS data and functions required to schedule activities within proposer and spacecraft constraints. The first section describes the SPSS inputs from upstream systems such as TRANS and SPIKE. This is followed by a detailed listing of the constraints that are evaluated by SPSS. We then describe the manner in which processing is broken down into distinct "planning" and "scheduling" stages in SPSS. Finally, the automated process of building an efficient calendar from a pool of observations is presented.

SPSS Inputs

The SPSS software is built around a Standard Query Language (SQL) relational database called the Project Management Data Base (PMDB). Data from several upstream systems generate the scheduling constraints that are consolidated in the PMDB. The following systems and the data they provide are described in this section:

- TRANS - transformed proposal instructions from RPS2
- POMS - crafted parallel observation data
- SPIKE - long range plan and relative constraints
- MOSS - moving target ephemeris
- PDB - spacecraft health and safety constraints
- GSSS - Guide Star data
- FDF - HST and TDRS ephemeris

With the exception of the systems at the Flight Dynamics Facility (FDF), which reside at the Goddard Space Flight Center, all of these systems operate at the Space Telescope Science Institute (STScI).

Figure 1 illustrates how these systems interface with SPSS via the PMDB and auxiliary files. This figure also shows the SPSS processing tasks described later in this paper.

Proposer Constraints

The science and engineering proposal specifications and constraints ingested by the upstream proposal submission system RPS2 are transformed into a hierarchical data structure for SPSS processing. This data structure is created by the front-end system called TRANS. The structure levels, from highest to lowest are

- Proposals
- Scheduling Units (SUs)
- Observation Sets (OBSETs)
- Alignments

- Exposures

Note that the concept of a "**visit**" exists in the HST front-end systems. SPSS has no visit concept. An exposure can be regarded as a shutter-open activity. An alignment is a set of one or more exposures for a specific pointing of HST. An observation set is a set of one or more alignments that normally use the same set of guide stars. A scheduling unit is a group of one or more related observation sets. A proposal contains one or more related scheduling units.

The Parallel Observation Matching System (POMS) uses pointing information from scheduled "**primary**" observations to craft "**parallel**" SUs based on parallel observing proposals. Parallel observations are scheduled at the same pointing but using different science instruments or detectors than primary observations. POMS receives a representation of a weekly schedule of primary observation activities from SPSS and generates parallel SU data that are loaded into the PMDB and scheduled onto the final flight calendar (see Figure 1).

Scheduling units can also be grouped into link sets. Link sets describe, often complex, relative timing or orientation constraints between sets of SUs that can span over long time periods (years). Proposer specified relative constraints are loaded into the PMDB via the SPIKE software.

Long Range Plan

The SPIKE system processes the proposal database and produces a Long Range Plan (LRP). The LRP spreads observations out evenly over an observing cycle and provides each scientist with an approximate time when their observation will be scheduled. The LRP consists of lists of windows for each Scheduling Unit. The nominal case would be one eight week long "**plan window**" for each SU, however windows could be shorter, or could cover the entire cycle. The SPIKE plan windows are stored in a front-end relational database called ASSIST, which are accessed directly and used by the SPSS software.

Long Range Planning Versus Short Term Scheduling

Note the distinction between long range planning in SPIKE and short term scheduling in SPSS. The set of plan windows provided by SPIKE for every scheduling unit is used by SPSS to identify those scheduling units that can be executed within a particular week. The number of scheduling units that meet this criteria typically exceed the maximum available observing time for the seven-day calendar. Calendars with an excess number of these candidate SUs are referred to as **oversubscribed**. Automated SPSS scheduling tools are used to maximize the efficiency of the final calendar in SPSS. When building a flight calendar, candidate SUs with plan windows that do not extend after the end of the seven-day calendar are given higher scheduling priority. Candidates that "fall-off" a calendar are planned for subsequent weeks.

Three primary factors distinguish long range planning performed by SPIKE and short term scheduling performed by SPSS. The first factor is the orbit uncertainty of the HST spacecraft due primarily to atmospheric drag. The orbital true anomaly of the spacecraft can only be determined a few weeks in advance with acceptable accuracy needed for detailed scheduling.

The second factor is computational workload. SPIKE effectively reduces the time span for many time consuming calculations to its generated plan windows. SPIKE predicts, to a reasonable accuracy, when during a cycle a particular visit can schedule, without having to schedule activities on a calendar. SPSS, on the other hand, can determine with absolute certainty when an SU may schedule – by physically attempting to schedule all of the SU's activities on a calendar. To compensate for SPIKE's plan window uncertainty, a verification process is in place to detect mismatches between the SPIKE and SPSS schedulability windows.

The third factor is the possible dependencies that exist between any two scheduling units on a flight calendar. With the exception of relative timing and orientation constraints, SPIKE models each observation independently. In SPSS, however, this property does not hold. The HST science instruments, for example, impose well-defined warm-up/cooldown state transition restrictions. These restrictions cause overall observation durations to vary depending on the placement relative to other observations in the timeline.

Moving Target Data

Moving target observations require target positional data that are not specified in proposals. Positional data for moving targets such as planets, moons, and asteroids are modeled using piecewise Chebychev polynomials and visibility windows calculated by the Moving Object Support System (MOSS).

A Target data structure maintained in the PMDB contains detailed fixed and moving target position and characteristics information.

Spacecraft Health and Safety Constraints

Spacecraft health and safety constraints such as sun, moon, and antisun avoidance regions, power and solar array constraints are maintained in the Project Data Base (PDB). Some of this data are made available to SPSS in the form

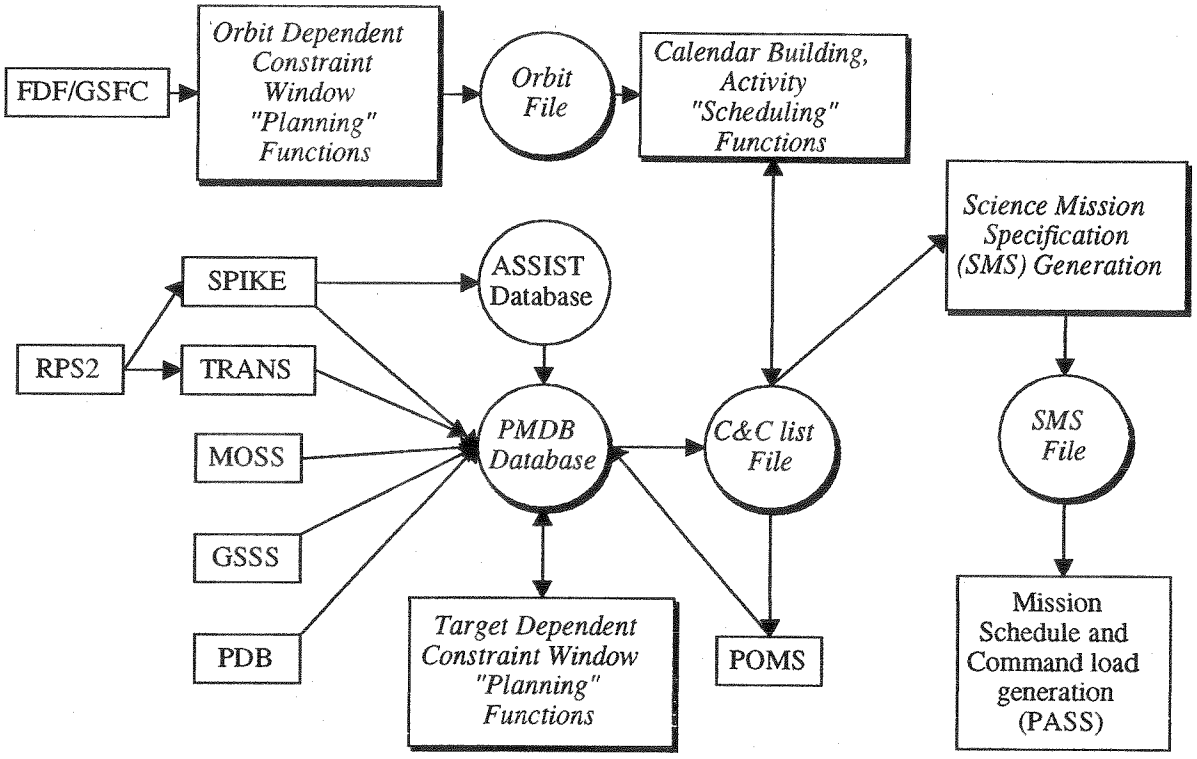


Figure 1: Diagram showing the high level interfaces between SPSS and the front-end proposal processing systems. Shadowed symbols with italicized text denote SPSS data and functions.

of namelists while others are loaded into relations in the PMDB. Science instrument operational constraints are directly maintained in the PMDB.

Guide Star Data

Guide Stars form another data structure. Single stars and pairs of stars for specific observation sets are determined by the Guide Star Selection System (GSSS) and stored in the PMDB. SPSS determines acquisition datasets for the guide star data. These datasets form spacecraft roll ranges for the observation set guide stars.

HST and TDRS Ephemerides

Predictive HST and TDRS (Tracking and Data Relay Satellite) position data are obtained from NASA's Flight Dynamics Facility (FDF) at the Goddard Space Flight Center (GSFC). In SPSS, the raw ephemeris is basically reduced to a set of non-linear equations that are used to estimate the HST and TDRS orbital elements as a function of time. This reduced data is stored in the PMDB.

SPSS Processing

SPSS Constraint Window Computation

As its name implies, SPSS processing is primarily composed of the "**planning**" phase, when scheduling windows are precomputed to reduce processing time, and the "**scheduling**" phase, during which spacecraft activities are scheduled on the nominal seven-day calendar.

Scheduling windows are defined as periods when some constraint is satisfied. In the planning phase, the constraints for which these windows can be computed are those that are either HST orbit dependent (e.g. SAA avoidance) or target dependent (e.g. sun avoidance), but not both. Scheduling windows for HST orbit dependent constraints are stored in an "Orbit file", and windows based on constraints that are solely target dependent are stored in the PMDB for every observation set. The orbit file windows are typically computed over a period of 10 weeks, which is the duration of HST and TDRS predictive ephemeris provided by the Flight Dynamics Facility (FDF)

at the Goddard Space Flight Center. Target dependent constraint windows are stored in the PMDB as computed over a time period specified by the user. During the scheduling phase, windows for constraints that are both HST and target position dependent (e.g. moving target earth avoidance) are computed on the fly.

Table 1 is a summary of the constraints that must be met when scheduling spacecraft activities. Note that constraint types generally fall under two categories or types: those that are ensure science integrity and those that maintain the health and safety of the HST.

Constraint	Constraint Type: H = Health and Safety S = Science Integrity	Subsystem responsible for computing windows	Computed during SPSS planning or scheduling stage?	SPSS Constraint Window Storage Area
Sun Avoidance	H	SPSS	Planning	PMDB
Anti-Sun Avoidance	H	SPSS	Planning	PMDB
HST Roll Orientation	H	SPSS	Planning	PMDB
Power : • Nominal Roll • Off-Normal • Solar Array Shadowing • Thermal	H	SPSS	Planning	PMDB
Moon Avoidance	S	SPSS	Planning	PMDB
Earth Avoidance	S	SPSS	Scheduling	On demand
South Atlantic Anomaly Avoidance	S	SPSS	Planning	Orbit File
HST in Earth Shadow	S	SPSS	Planning	Orbit File
Moving Target Visibility	S	MOSS	N/A	PMDB
Phase Windows	S	PMDB	Planning	PMDB
Science Instrument Parallel Usage	S	SPSS	scheduling	On demand
Minimum/Maximum Separation	S	SPSS	Scheduling	On demand
Maximum Interruption	S	SPSS	Scheduling	On demand
Fine Guidance Sensor Constraints: • Earth Avoidance • SAA Avoidance • Guide Star Constraints (Roll Angle)	 S S S	 SPSS SPSS SPSS	 Scheduling Planning Planning	 On demand PMDB PMDB
Fixed Head Star Tracker Constraints: • Moon Avoidance • Earth Avoidance	S S	 SPSS SPSS	 Scheduling Scheduling	 On demand On demand
User Specified (US) Windows	S	SPSS	Scheduling	On demand
Timing Link Set	S	SPSS	Scheduling	On demand
Orientation Link Set	S	SPSS	Scheduling	On demand
Plan Windows	S	SPIKE	N/A	PMDB

Table 1: Summary of SPSS constraint types. Note that pre-computed windows that are stored in the PMDB are eventually copied to the C&C list for use in scheduling. Windows that are calculated "On demand" are stored in internal memory as needed. For more detailed information on these constraints, see the on-line references listed in this paper.

Calendar Building and Activity Scheduling

For a given weekly schedule, SU data are copied from the PMDB to a file called a Candidate and Calendar List or "C&C List" file. This file contains a list of candidate SUs and a calendar of scheduled activities that support the observations specified in each SU. The final calendar contains the detailed schedule of all spacecraft activities for the time span. A file is used instead of direct SQL manipulations to improve performance. This file is essentially a snapshot of what is in the PMDB at the start of a calendar building week. If necessary, the C&C list can be updated with new data in the PMDB (e.g. if proposal specifications change) on demand.

Scheduling constraints apply to virtually every activity scheduled. The scheduling process determines what constraints apply to a given activity, either by rules built into SPSS, or by database specified flags. After determining which constraints apply for a given activity, windows are computed (or retrieved, if pre-computed) for times where the constraints are satisfied, and the activity is scheduled at the earliest opportunity within the window. If a constraint cannot be satisfied, then the scheduling fails, otherwise the activity is delayed until the constraint has cleared.

SPSS schedules all of the current HST science instruments, Fine Guidance Sensors (FGSs), Gyros, Fixed Head Star Trackers (FHSTs), and other spacecraft hardware. The spacecraft activities that SPSS schedules include:

- Vehicle Maneuvers : Slews, Small Angle Maneuvers, and Track 51 Linear Scans (moving target tracking)
- Pointing Control System : Guide Star Acquisitions, Guide Star Reacquisitions, Guide Star Handoffs
- Target Acquisitions
- Fixed Head Star Tracker Updates: Delay Mode Roll Updates, Maneuver Mode Full Updates
- Science Instrument Reconfigurations
- Science Activities: Data Collection, Calibration, Interleaver Science, Parallel Science
- Tracking & Data Relay Satellite Contacts
- Tape Recorder Usage

As mentioned earlier, the SPSS software must schedule observations under a large number of constraints. For example, some instruments are sensitive to the South Atlantic Anomaly (SAA). SPSS determines time spans of HST passage through various SAA models and schedules observations to avoid these periods. Observations must avoid the bright and dark Earth by different avoidance angles. SPSS must model the Earth's terminator and determine avoidance windows depending on the instantaneous Sun, HST, and fixed/moving target positions.

Various automatic, manual, and graphical are used to schedule the observations on the calendar. The observations are scheduled to maximize the scientific return from HST.

The scheduling unit (SU) is the basic unit of scheduling within SPSS. That is, a scheduler will specify that a scheduling unit is to be added to a calendar. To be considered successful, every supporting activity must be scheduled constraint free. The calendar builder is initially provided with an oversubscribed list of scheduling units with plan windows (from SPIKE's long range plan) that overlap the seven-day calendar. Oversubscription poses the following problem: which of the scheduling units are to be placed on the calendar, and in what order so as to maximize "efficiency"?

An automated scheduling tool that implements a heuristic based greedy search algorithm is used to arrive at a best ordering of scheduling units. The problem can be modeled as the need to search for the best possible path down an n-ary tree, where each node represents a scheduling unit that is scheduled in the best possible place (not necessarily in chronological order) on an updated calendar.

The calendar is built by starting at the root node where a heuristic is applied to determine which path to take to the next level in the tree. Once a path is determined, the scheduling unit is permanently scheduled to create an updated calendar, which serves as input to the selection process down subsequent levels in the tree. A complete solution is obtained when a leaf is reached or when the heuristic is unable to make a valid selection.

The heuristic consists of two parts. The first part involves the ranking of each scheduling unit based on several factors, which includes a user specified priority factor and "difficulty to schedule" adjustments. In the second part, the top N-ranked scheduling units are added alone on the input calendar, where N is a user-specified parameter. A new score is computed for each scheduling unit/place combination. The scheduling unit with the highest place score is then added permanently on the calendar.

The greedy search has its shortcomings in that the resulting calendars have required manual repair to accommodate tight constraints. This algorithm can likely be improved by including knowledge of SU schedulability windows – i.e. windows where an SU will schedule on an empty calendar.

SPSS Outputs and Post-Calendar Processing

When the calendar is completed, other SPSS tools update the PMDB with basic information about the scheduled

SUs. Finally, a Science Mission Schedule (SMS) file is generated from the calendar and data in the PMDB. An SMS is sequence of detailed spacecraft commands. Another subsystem, PASS, verifies the SMS, inserts health and safety activities, and generates spacecraft ready command loads.

SPSS Software and Hardware Characteristics

The SPSS software consists of nearly one million physical lines of F77 and C/C++ code in about one hundred executable programs. Most of these programs are supported on both AXP/VMS and Sparc/Solaris platforms.

Summary

SPSS has been used operationally in the planning and scheduling of HST activities for the past seven years. Several modifications have been made to SPSS during this period to accommodate new instruments as well as new spacecraft capabilities and constraints. The system has been proven to adapt well to these changes.

References

More detailed documentation can be found on the World Wide Web at the following URLs:

http://www.pst.stsci.edu/~samson/overview_doc.cgi
http://www.pst.stsci.edu/~samson/windows_doc.cgi

Simulation and Planning for Food Production Scheduling

Jeff Schneider and Andrew Moore
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
schneide@cs.cmu.edu,awm@cs.cmu.edu

Introduction

Schenley Park Research's PlanIt software is currently being used by a major U.S. food manufacturer to schedule production in one of their factories. An overview of the system is shown in fig. 1 and a sample schedule is shown in fig. 2.

The planner is presented with a problem specification and formulates a schedule to meet the constraints while maximizing profit. A specific problem instance is given as a set of inventory profiles for approximately 20 products. The inventory profiles reflect the current stock of each product and the estimated demand for that product over the period covered by the plan. The goal is to generate a cost optimal production schedule which prevents the inventory of any product from dropping below a danger threshold. This must be done subject to constraints on which products can be produced in combination with other products.

In order to evaluate potential schedules, the planner relies on a simulation of the factory and a knowledge base representing costs and constraints. The user has the opportunity manually edit the plan. The resulting plan is sent off for execution, which is not directly controlled by the software. The results of the execution, which often vary significantly from the expectations of the original schedule, are automatically fed back to the system for re-planning. A typical schedule covers a period of 1 to 4 months and takes the planner 1-12 hours to produce. Replanning is done anywhere from once a day to once a week.

A sample schedule is shown in fig. 2. There are several packagers available to handle about 20 different food products. The schedule shows which products they should be set to produce over time. Additional complexity comes from the constraints and interactions not made explicit by this representation of the schedule. There are several other stages of processing which occur before and after the packaging stage shown. It is not necessary to explicitly schedule all the other operations, but they must be represented in the knowledge base because they constrain what products can be produced simultaneously. Additionally, the rate at which a particular packager produces a particular product is

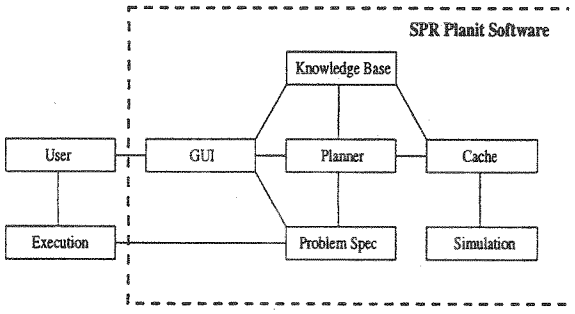


Figure 1: Overview of planning and execution process for production scheduling

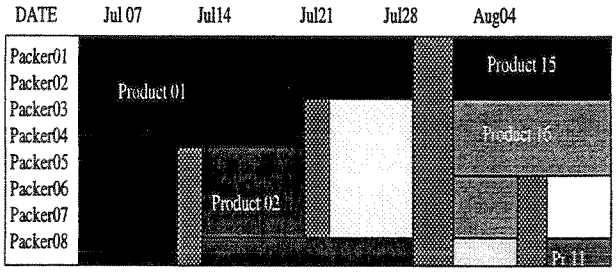


Figure 2: A simplified example of a schedule produced by SPR PlanIt for a single production line. The crosshatching between products reflect the amount of time required to changeover between the products.

dependent upon what the other packagers are doing at the time. The time required to reconfigure them to produce a new product is also a complex function of the current setup of the factory.

Simulation and Caching

The simulation and caching modules are crucial to the solution of this scheduling problem. First, simulation is required to accurately determine how much of each product will be produced with a particular factory configuration. Standard linear models of production are insufficient. The performance of the factory components is highly stochastic, so simulations must be run

for a significant amount of time to be accurate. This could make planning infeasible because of the computation required. In order to solve that problem, all simulation results are cached. It is impractical to cache the results for every possible factory configuration, but many of them are never useful. After several planning runs, a cache of 10,000 to 50,000 configurations is generated, which covers almost all the ones the planner is likely to consider.

Planner

The planner uses a heuristic combination of several common search and optimization algorithms including greedy search, hill climbing, simulated annealing, and linear programming. Many of these algorithms are used in several different ways to optimize different aspects of plans. For example, a linear program that optimally sets the length of time each product is run on each resource, is run as a subroutine to a hill climber that changes which products are scheduled for each resource. The talk will discuss the how these algorithms are combined and their pros and cons.

Discussion

There are several properties that make this scheduling problem more difficult than typical formulations such as job shop scheduling. Both constraint satisfaction and cost optimization play a strong part in the planning. algorithms and representations that focus too much on either one are insufficient. Expensive, nonlinear simulations are the only way to accurately model the production process. This restricts both the time available for planning and excludes algorithms that rely on simple production models. Finally, there are no single deadlines for production requirements. The requirement is an inventory curve that must be considered at all points in time. Because of the complexity of this problem, we believe the techniques developed for it can be applied to a variety of space related applications. The scheduling of scientific sensors with interacting resource requirements on a spacecraft is one obvious possibility and we hope to discuss many more at the workshop.

The biggest obstacle during the development of this system was that the factory specifications were a moving target. The capabilities of the factory were changing on nearly a monthly basis. Frequently, new products were being added, or poor sellers were being removed. Occasionally, entire production lines were added, removed, or redesigned. The smallest changes called for updates to the knowledge base and the problem specifications. Larger changes required modification of the simulation and caching algorithms. Some changes also called for re-engineering the algorithms used in the planner. The issue of a constantly changing factory remains the biggest problem in the fielded application. The same problems arise in the goal of rapidly deploying the software to a series of new factories, or in using the software to do what-if analysis on potential factory designs.

Small changes are handled well in the current system. A typical example is the addition of a new product. This can be accomplished by using the GUI to edit the knowledge base. The end users perform that kind of change themselves.

Larger changes generally require modifications to the simulator. A typical example is a change in the distribution system that divides the raw product between the resources. Deploying the system in a new factory will often require the implementation of a new simulator as well. Since this is done in C code in the present system, it can be time consuming and require more expertise than factories have available on the factory floor. One method for dealing with this problem is to use simulation suites that are easy for non-programmers to use. Better simulation tools help, but it is still likely that coping with significant factory changes will be expensive.

An alternative is to expand the sources of information that can be used to fill the cache. In the current system, all information in the cache comes from simulation, but there are other potential sources. One is the knowledge base. In simple cases, approximations to the production statistics can be written down in closed form and stored in the knowledge base. Another source of information is the results of executions. A separate module could be included to learn the behavior of the factory by monitoring it. In addition to storing information, the cache would be responsible for combining the information from various sources depending on how reliable each is. In the best case, simple rules could be added to the knowledge base which would be good enough to allow planning while the learning system came up to speed and/or a simulation is constructed for a new factory.

Finally, some changes affect the performance of the planning algorithms. It requires a significant amount of time from an expert to tune the planner. A big improvement would be to automate the process of testing and tuning the planning algorithms. This is a research problem that requires all the algorithms to be put into a common framework such that they can be slotted in and out of the planner, as well as parameterizing them in a way that can be searched efficiently.

ASPEN: EO-1 Mission Activity Planning Made Easy

Rob Sherwood, Anita Govindjee, David Yan,
Gregg Rabideau, Steve Chien, Alex Fukunaga
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California 91109
aspen@aig.jpl.nasa.gov

ABSTRACT

This paper describes the application of an automated planning and scheduling system to the NASA Earth Orbiting 1 (EO-1) mission. The planning system, ASPEN, is used to autonomously schedule the daily activities of the satellite. The satellite and operations constraints are encoded within a software model used by the planner. This paper includes a description of the planning system and the associated modeling language. We then discuss how we encoded the EO-1 spacecraft operations with the modeling language. We conclude with a description of the end-to-end planning system as we envision it for EO-1.

INTRODUCTION

Automated planning/scheduling technologies show great promise in reducing operations costs by increasing autonomy of EO-1 mission operations. The Artificial Intelligence (AI) Group at the Jet Propulsion Laboratory[1] has been working on a system called ASPEN (A Scheduling and Planning Environment). ASPEN [Fukunaga et al. 1997] is a modular, reconfigurable application framework based on AI techniques [Allen et al. 1990, Zweben & Fox 1994], which is capable of supporting a variety of planning and scheduling applications (similar to [Smith et al. 1996]). The primary application area for ASPEN is the spacecraft operations domain.

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

This paper appears in the Working Notes of the NASA Workshop on Planning and Scheduling for Space.

[1] For an overview of Artificial Intelligence work at JPL, see [Chien et al. 1997].

EO-1 [Speer et al, 1997] is an Earth imaging satellite to be launched in May 1999. The science payload on EO-1 is an advanced multi-spectral imaging device. Mission operations on EO-1 consist of managing spacecraft operability constraints such as power, thermal, pointing, buffers, consumables, and telecommunications. EO-1 science goals involve imaging of specific targets within particular observation parameters. Of particular difficulty is managing the downlinks since the amount of data generated by the imaging device is quite large and ground contacts are limited. In addition, because science targets for EO-1 are based on short-term cloud predictions, schedules must be generated daily.

Planning and scheduling spacecraft operations involves generating a sequence of low-level spacecraft commands from a set of high-level science and engineering goals. ASPEN encodes spacecraft operability constraints, flight rules, spacecraft hardware models, science experiment goals, and operations procedures to allow for automated generation of low-level spacecraft sequences. By automating the command sequence generation process and by encapsulating the operations specific knowledge, ASPEN will enable EO-1 spacecraft commanding by a small operations team and thereby reduce costs.

ASPEN

ASPEN is an object-oriented system that provides a reusable set of software components that implements the elements commonly found in complex planning/scheduling systems. These include:

- An expressive constraint modeling language to allow the user to naturally define the application domain;
- A constraint management system for representing and maintaining spacecraft

```
1    Activity ALI_data_take {
2      Fixed fi;
3      Tracking tr;
4      Duration = [1,60];
5      Constraint =
6              starts_after end of SAD_changer with (fi->sad1) by [100,300],
7              ends_before start of SAD_changer with (tr->sad1) by [16,16],
8              Contains both of SAD_user with (fi->ap1) by [4,4,0,10],
9              Contained_by both of SAD_user with (fi->sad1) by [300,300,1,16];
10     Subactivities = ALI_user_data, ALI_dark_count;
11     Simple_reservations = processor, array_power = 80;
12   };
13
14   Activity SAD_changer {
15     Sad_mode sad1;
16     Simple_reservations = solar_array change_to sad1,
17                 aperture must_be open;
18   };
```

Figure 1 Activity Example

operability and resource constraints, as well as activity requirements;

- A temporal reasoning system for expressing and maintaining temporal constraints; and
- A graphical interface for visualizing plans/schedules (for use in mixed-initiative systems in which the problem solving process is interactive).

The central data structure in ASPEN is an activity. An activity represents an action or step in a plan/schedule. An activity has a start time, an end time, and duration. Activities can use one or more resources. For more details on ASPEN, see [Fukunaga *et al* 1997].

MODELING LANGUAGE

The ASPEN modeling language allows the user to define activities, resources, and states as described above. A domain model is input at start-up time, so modifications can be made to the model without requiring ASPEN to be recompiled. The modeling language has a simple syntax, which can easily be used by spacecraft mission operations personnel to create a model. Each spacecraft model is comprised of several plain-text files, which define and instantiate activities, resources, and states.

Activities

As previously mentioned, activities are the central data structure of ASPEN. An activity is a data structure that performs a specific function. The example in Figure 1 includes an instrument data take activity and a solar array drive state change activity. These examples will be used to explain the components of an activity.

An activity is defined in line 1 and 14 of Figure 1. The definition includes the name followed by a pair of braces and a semi-colon similar to the C language syntax. These are the only required components of an activity definition. Once the activity is defined, it can be instantiated in the initial state file. Generally, this instantiation will consist of just the activity name followed by the instantiated name and a pair of braces. Many of the components below that are specified as ranges can be fixed to specific values in the activity instantiation.

Parameters are generally used to pass values to activities or reservations. Lines 2 and 3 contain parameters defined elsewhere in a parameters file. In this case, they are constants that represent state names. Parameters can also be passed into activities from higher level activities (parent activities). Line 15 contains an example of a parameter that is passed into an activity. The parameter sad_mode is an enumerated type variable that contains the list of states for a solar array drive. Any one of the states can be passed into the SAD_changer activity when called from a parent activity.

The duration of an activity is given as a range [x, y], a list {a, b, c, d} or a constant. Line 4 defines the duration as a range of 1-60 seconds. The time scale of the spacecraft mission planning can also be specified. All ranges within ASPEN can be specified from zero to *infinity*. If a range is given for the duration, ASPEN will have more flexibility in considering different schedules. This can result in better-optimized schedules.

Constraints are temporal constraints an activity must satisfy with respect to the (parent) activity in which the constraint is defined. There are six types of constraints: starts_before, starts_after, ends_before, ends_after, contains both of, and contained by. The first four constraint types include a time range and a temporal relationship to the start of or end of the activity in question. For example, on line 6 in Figure 1, the ALI_data_take activity must start after the end of the SAD_changer activity by 100-300 seconds. This constraint tells the scheduler that the SAD_changer activity must be completed at least 100 seconds before the ALI_data_take activity starts. Using the same method, the start or end time of any activity can be specified relative to the start or end time of the parent activity. If the time duration is specified as [0,0], the start or end times will coincide exactly.

The [contains both of] constraint is used for activities that fall within the parent activity. This constraint definition combines a starts_before start of and an ends_after end of pair of constraints. For example, line 8 defines a constraint for the SAD_user activity that is contained within parent activity ALI_data_take. The first two and last two numbers in the constraint represent ranges of time, which separate the start times between the two activities and the end times between the two activities. SAD_user must start exactly four seconds (4,4) after the start of ALI_data_take but the end time can coincide with the end time of ALI_data_take or up to 10 seconds (0,10) earlier. This relationship is graphically represented in Figure 2.

The [contained_by both of] constraint is used for activities that are the same size or larger than the parent activity. For example, line 9 defines a constraint for activity SAD_user that starts exactly 300 seconds before the start of and ends 1-16 seconds after the end of activity ALI_data_take.

Subactivities are activities that can be scheduled any time within the parent activity subject to resource constraints within the subactivity. Subactivities are similar to the constraint-defined activities above without the exact temporal relationship between the parent and subactivities. For example, line 10 defines subactivities ALI_user_data and ALI_dark_count. These activities must fall within the temporal range of the parent activity ALI_data_take.

Reservations are used to reserve a portion of a resource or state for the duration of the activity. There are two types of resource reservations: atomic and non-atomic. Line 11 of Figure 1 contains examples of an atomic reservation (processor) and a non-atomic reservation (array_power). The processor reservation reserves the processor for the duration of the activity. No other activities can use the processor during this time. The array_power reservation uses 80 units of array_power for the duration of the activity. If the array_power were a depletable resource, the 80 units would be reserved from the start of the activity until the end of the planning horizon.

State reservations either change the state of a state variable or reserve a state for the duration of an activity. Line 16 of Figure 1 changes the state of

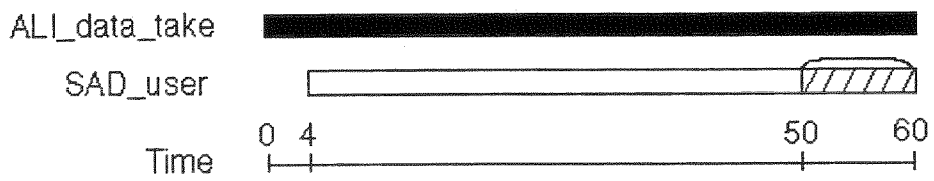


Figure 2 Constraint Relationship: contains both of

the SAD state variable to the value of parameter sad1. Line 17 of Figure 1 reserves the "open" state of the aperture state variable for the duration of the activity. If the aperture state variable was in a state other than "open" prior to this activity, the state is changed to "open" by the reservation.

Resources

Resources are items that can be scheduled. There are four types of resources: atomic, concurrency, depletable, and non-depletable. Atomic resources are physical devices that can only be used (reserved) by one activity at a time. Examples of atomic resources include: science instrument, star tracker, reaction wheel, or CPU. Concurrency resources are similar to atomic except they must be made available to the activity before they are reserved. An example would be a telecommunications downlink pass. The telecommunications station would have to be made available before the spacecraft could initiate a downlink. Non-depletable resources are resources that can used by more than one activity at a time and do not need to be replenished. Each activity can use a different quantity of the resource. Examples include solar array power and 1773 bus. Depletable resources are similar to non-depletable except that their capacity is diminished after use. In some cases their capacity can be replenished (battery energy, memory capacity) and in other cases it cannot (fuel). A summary of the four types of resources is presented in Table 1.

The four types of resources are defined in lines 1, 6, 12, and 18 of Figure 3. The definition includes the name followed by a pair of braces and a semi-colon similar to the C language syntax. The types is one of: atomic, concurrency, depletable and non-depletable. The name and type are the only required components of a resource definition. Once the resource is defined, it can be instantiated in the initial state file. Generally, this instantiation will consist of just the resource name followed by the instantiated name and a pair of braces. Note: concurrency resources are not yet implemented.

The capacity of a resource can be specified as a constant, list or range. A range would be used if several similar resources with specific capacities were defined when the resources were instantiated.

Resource Type	Properties
Atomic	Always available when not in use, only 1 user at a time Ex: science instrument, star tracker, reaction wheel, cpu
Concurrency	Only available when made available, only 1 user at a time Ex: telecommunications downlink pass
Non-depletable	Always available when not in use, many users can use different quantities Ex: solar array power and 1773 bus
Depletable	Capacity is diminished after use, may or may not be replenished by another activity Ex: battery energy, memory capacity, fuel

Table 1 Resource Types

```
1   Resource ALI {
2     Type = atomic;
3     Capacity = 1;
4   };
5
6   Resource Solar_array {
7     Type = non_depletable;
8     Capacity = 600; // watts
9     Min_capacity = 0;
10  };
11
12  Resource warp_storage {
13    Type = depletable;
14    Capacity = 40000;  // Mbits
15    Min_capacity = 0;
16  };
17
18  Resource Propellant {
19    Type = depletable;
20    Capacity = 15;  // 15 kg
21    Min_capacity = 0;
22  };
```

Figure 3 Resource Examples

An atomic resource has a unit capacity and does not have to be explicitly set such as on line 3 of Figure 3. Depletable and non-depletable resources definitions can contain a minimum capacity such as in lines 9, 15, and 21 of Figure 3.

```
1    State_variable ALI_sv {
2            states = ( "data", "standby", "idle", "off" );
3            transitions = ( "standby"->"data", "data"->"standby", "idle"->"standby",
                                    "standby"->"idle","off"->"idle", "idle->off");
4            default_state = "idle";
5    };
6
7    State_variable aperture_sv {
8            states = ( "open", "closed");
9            transitions = ( "open"->"closed", "closed"->"open" );
10           default_state = "closed";
11   };
```

Figure 4 State Variable Examples

States

A device, subsystem, or system may be represented by a state variable that gives information about its current operation. The state variable contains the current state, which is defined as an enumerated type. Some examples of possible states are: on, off, open, closed, record, playback, standby or idle. States can be reserved or changed by activities. A state variable must equal some state at every time. At the beginning of a planning horizon, this state is just the default state. Figure 4 contains two examples of state variable definitions.

A state variable is defined in lines 1 and 7 of Figure 4. The definition includes the name followed by a pair of braces and a semi-colon similar to the C language syntax. Lines 2 and 8 contain a list of the states the state variable can contain. The default state must be defined and must be one of the states in the list. Once the state variable is defined, it can be instantiated in the initial state file.

The allowable state transitions between states can be indicated using the 'transitions' keyword with a forward (->) arrow, a bi-directional arrow (<->), or with the 'all' keyword (e.g., all<->all).

Parameters

The ASPEN modeling language includes parameters, which are variables or constants. Parameters can consist of integers, strings, floats, or lists. Parameters can be defined as enumerated types for a list of states in a state variable. Ranges of values can also be used. Some examples are:

- parameter string ALI_mode { domain = ("data", "idle", "standby", "off"); };
- parameter int warprange { domain = [1,40960]; };

In the first example, the ALI_mode parameter can take on any of the four values in the list. In the second example, the warprange parameter can be any integer in the indicated range from 1 to 40960.

EO-1 MODEL

EO-1 is an Earth imaging satellite that is part of the New Millennium Program of technology validation missions. The NASA Goddard Space Flight Center is responsible for project management. The purpose of EO-1 is to validate new technologies that can be used on future Landsat class Earth remote sensing missions. In fact, EO-1 will be flying in formation one minute behind Landsat-7, with the goal of imaging as many of the same targets as possible. EO-1 will be using the Landsat 7 daily scene list as an input file of potential EO-1 targets.

The main activity in EO-1 operations is the Advanced Land Imager (ALI) data take. The ALI instrument contains six separate detectors that output data simultaneously. One image takes a total of 24 seconds and consumes about 19 gigabits of data in the solid state recorder (WARP). Because the capacity of the WARP is only 40 Gbits, it is important to plan the data takes and downlinks to maximize the amount of data returned. Due to limited amount of downlink

time available, only four data takes per day can be taken. Data takes can be prioritized based on the following parameters:

- Cloud cover over the region to be imaged
- Sun angle at the region to be imaged
- Ability to return the data before overflowing the WARP recorder
- Images coinciding with Landsat 7 images
- Imaging of scientifically interesting areas

Each EO-1 data take has several conditions that must be satisfied before and after the data take occurs. These conditions are listed below:

Before:

- Change the ACS mode to science
- Change the solar array to a fixed orientation
- Open the ALI aperture
- Change the data rate to high rate mode

After:

- Close the ALI aperture
- Take one second of calibration data
- Change the ACS, solar array, and data rate modes back to the previous values

Each of these conditions is modeled as temporal constraints in the ALI data take activity. The data take activity itself is only a 24-second activity. The constraints on the activity span a period of 5 minutes before and one minute after the bounds of the activity. The constraints on the activity could have been modeled as activity decompositions. The reason we chose to model these activities as constraints is to ensure they would move with the parent activity if the parent activity were moved. The data take activity breaks down into 14 separate activities as listed in Figure 5.

The ALI must be calibrated by viewing the sun or the moon regularly. The sun calibration involves pointing at the sun and changing the aperture filter several times. The moon calibration points at the lunar limb and pans across the moon using each of the detectors. Similar to the data take activities, the calibrations involve several constraints. The calibration activities and constraints are listed in Figure 6.

ALI Scene Collection	
ALI_data_take	aperture_changer
ALI_user_data	engdata_user
ALI_user_standby	engdata_changer
ALI_changer	ACS_user
SAD_user	ACS_changer
SAD_changer	cloud_cover_changer
aperture_user	sun_angle_changer

Figure 5 EO-1 Science Activities

EO-1 communication activities are modeled as follows:

1. An input file gives the times at which the ground station is in view of the satellite.
2. The in view times are converted into a state variable with the value 'inview' or 'outview.'
3. The planner chooses communication links during these in view times.
4. The communication link is broken down into uplinks (if required) and downlinks.

The EO-1 model also includes initialization activities for power, propellant, and memory. These activities are used to keep track of consumable resources from the previous planning period.

A keyword 'command' is used for activities that represent an EO-1 spacecraft command. When the command keyword is included in the activity definition, along with the command name, the spacecraft command output file will include a time tagged command for that activity.

The EO-1 spacecraft resources are modeled as either depletable or non-depletable. It was not necessary to model every physical device on EO-1 because many devices consumed a constant power and did not interact with any spacecraft activities. The power of these devices is included in the power_init activity. The resources that are modeled are listed in Figure 7.

ALI Calibrations
ALI_sun_calibration
slew_to_sun
aper_test_changer
ALI_moon_calibration
moon_cal_ms_pan
slew_to_moon
ramp_up_pitch_slew
ramp_down_pitch_slew
roll_to_next_position

Figure 6 EO-1 Calibration Activities

Resources	
Non-Depletable	**Depletable**
ALI	Battery
S_band_Receiver	Warp_storage
Transponders	Propellant
solar_array	
ACDSE	
Warp	
Processor	
Bus_1773	
Cat_bed_heater	
WFF	
DSN	

Figure 7 EO-1 Resources

The EO-1 ASPEN model has ten different state variables which are listed in Figure 8. Most of these state variables are used to represent the state of a spacecraft resource. The states are used in activities that require a resource to be in a particular state. These requirements are specified in the reservations of the activity. For example, the EO-1 data take activity requires the WARP state variable to be in record mode during the period of imaging. This requirement ensures that the data is being recorded during the imaging operation. Activities are defined that either change or use a particular state of a state variable. These activities usually contain a command keyword that corresponds to an EO-1 spacecraft command.

State Variables	
Variable	***Possible States***
ALI_sv	data, standby, idle, off
SAD_sv	off, tracking, fixed
aperture_sv	open, closed
aperture_test_sv	small, med, large, blank
engdata_sv	high, low
ACS_sv	nadir, low_jitter, standby, safe, orbit_adjust,
WARP_sv	off, idle, record, playback
Cloud_Cover_sv	low, med_low, med, med_high, high, none
Sun_Angle_sv	low, med, high, none
WFF_inview_sv	in, out

Figure 8 EO-1 State Variables

Creating the EO-1 Model

The modeling language has been designed such that it can model a physical spacecraft system directly. It is a descriptive language that allows an engineer to directly represent the physical spacecraft information into the model. In fact, the EO-1 model was created by an engineer (first author, Rob Sherwood) who had no knowledge of the software or its algorithms and procedures. He successfully created the model by simply taking the EO-1 spacecraft information and putting it into the modeling language syntax. This process took three weeks. Another similar model for the Spacecraft Interferometry Mission took less than two days.

The modeling language is flexible and allows for different ways of representing the same information. Therefore, there is no *one* correct model for a given spacecraft. The EO-1 model is constrained to have certain state variables, for example, as determined by the mission, but, on the other hand, could have different ways of representing constraints between activities.

END-TO-END PLANNING SYSTEM

The goal of this EO-1 work is to produce an automated on-board planning system for spacecraft commanding of the EO-1 satellite. The system would be validated after launch on the ground. As a ground based planner, the inputs to ASPEN include:

- Landsat-7 cloud cover and sun angle predictions.
- Current power, propellant, and memory levels.
- Sun, moon, and sky calibration requests.
- Ground station view files.
- Maneuver requests.

The output of the ground based validation of the planner would be a text list of time tagged commands that would be translated into binary spacecraft command by the ground system load generation utility. This utility is already built into the EO-1 ground system.

The on-board planning system would require uploads of the ground station view files and maneuver requests. The cloud cover could be obtained by using the ALI science instrument to examine the clouds before a scene. After the image is taken, the cloud data would be analyzed to determine if the scene should be saved and

downlinked. Clouded scenes would be erased from the WARP and a new scene would be planned to take its place.

EO-1 Model in Action

Generating EO-1 mission operations schedules is a fast process. Given a set of EO-1 requests, ASPEN will generate a conflict-free schedule within the order of a few minutes for lengthy schedules, and within seconds for simpler schedules. For example, for 162 EO-1 activities, it takes ASPEN 3.53 seconds (on a SUN Ultra-2) to produce a conflict-free schedule. There are no EO-1 schedules that take more than a minute to schedule, but with other spacecraft models with more activities and lengthier schedules, we have seen a maximum of five minutes to produce a conflict-free schedule.

In addition to having the activity requests specified in advance, the user can make changes to the schedule from the GUI as needed. For example, the user could add an ALI_data_take activity. If this caused conflicts in the schedule, then ASPEN would resolve the conflicts. This whole process takes seconds to execute. For example, with the EO-1 model, if we add three ALI_data_take activities in the GUI (randomly placed), this causes 34 conflicts. Resolving all conflicts, and producing a conflict-free schedule takes 1.54 seconds (on a SUN Ultra-2). This means that it is solving approximately 17 conflicts per second. (Adding just one data-take activity causes a large number of conflicts because of the constraints between activities and the states required by different activities.)

Currently, activities can be given a particular score, and high-level preferences (such as *resource max usage*) can be indicated which also determine scores for activities. The generated schedule is then given a score based on the activities' scores. Using this score, the user can then choose one generated schedule over another. We are presently working on an algorithm that will automatically optimize schedules.

CONCLUSIONS

Modeling EO-1 mission operations in ASPEN is easy and compact. The entire EO-1 model consisting of the activities, parameters, reservations, resources, and state variables as described above, is represented in approximately 700 lines (in plain text files) which also includes comments and headers. The simplicity of the modeling language will allow the operations personnel to easily change the model when needed. The changes will not require a recompile of the code.

We have successfully modeled EO-1 mission operation activities with ASPEN. We have created a model which encapsulates information about: data takes, calibration activities, maneuvers, uplinks, downlinks, validation activities, cloud cover and sun angle states, and initialization activities of power, propellant, and data storage. Using this model with ASPEN will enable EO-1 to function with a very small operations team.

REFERENCES

J. Allen, J. Hendler, and A. Tate 1990. *Readings in Planning*. Morgan Kaufmann.

S. Chien, D. Decoste, R. Doyle, and P. Stolorz 1997. Making an Impact: Artificial Intelligence at the Jet Propulsion Laboratory. *AI Magazine*, 18(1):103-122.

A. Fukunaga, G. Rabideau, S. Chien, and D. Yan 1997. ASPEN: A Framework for Automated Planning and Scheduling of Spacecraft Control and Operations. In *Proceedings of the International Symposium on AI, Robotics and Automation in Space (i-SAIRAS)*, Tokyo, Japan.

S.F. Smith, O. Lassila, and M. Becker 1996. Configurable mixed-initiative systems for planning and scheduling. In A. Tate, editor, *Advanced Planning Technology*. AAAI Press.

D. Speer, P. Hestness, M. Perry, and B. Stabnow, The New Millennium Program EO-1 Mission and Spacecraft Design Concept, In Proceedings of the IEEE Aerospace Conference, v. 4, pp. 207-227, Snowmass, CO, 1997.

M. Zweben and M. Fox 1994. *Intelligent Scheduling*. Morgan Kaufmann.

Design for X (DFX): Operations Characteristics Spacecraft Design Analysis Tool

Robert Sherwood, Steve Chien, Gregg Rabideau, Tobias Mann

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, M/S 525-3660
Pasadena, CA 91109-8099

Abstract

This paper describes a system to assist in evaluating operations characteristics of preliminary spacecraft designs. This system, called *Design for X* (DFX) requires: a set of operations goals (science and engineering goals and constraints), a model of the spacecraft, a model of spacecraft activities, and a set of scoring functions which evaluate the utility of different operations activities (e.g., science value). DFX uses this modeling information and artificial intelligence based planning and scheduling techniques to produce a high level activity plan that is then scored using the provided functions. This process automates the evaluation of spacecraft designs which has several benefits: improved science return due to optimized sciencecraft, improved spacecraft operability due to more accurate margins and interactions analysis, decreased project (e.g., budget, schedule) risk from using rapid prototyping and analysis of designs. In addition, such a tool could assist in performing more methodical trades analyses and the tool could be used for impartial analyses of science Announcements of Opportunity (AO)'s.

1. Introduction

Spacecraft design is a challenging task that is both knowledge and labor intensive. Spacecraft and mission designers must balance numerous competing constraints on mass, power, cost, volume, and other systems level interactions. In addition, many of the desired attributes of the end spacecraft cannot be directly evaluated as spacecraft parameters. For example, enhancing the science return is a desired goal. However, it is often difficult to evaluate the quantitative influence direct design parameters such as memory, or pointing speed would have on science return. Another example, ease of operation is a desired characteristic of a spacecraft. While many specific design parameters would make a spacecraft easier to operate (more power, more memory, less thermal constraints, etc.), knowing the exact impact of modifications on operability is more difficult. For example, which resources are closest to their margins during nominal science scenarios? Which resource usages are most sensitive to changes in the estimated slew times? How does downlink rate affect the memory margins? The Design for X (DFX) tool is intended to assist in answering these questions.

The DFX system is targeted at providing a "what-if?" capability for evaluating spacecraft designs. The intended method of usage for the DFX tool is shown below in Figure 1. The DFX system requires as input:

- A candidate spacecraft design (and operations constraints, models, etc.)
- A set of engineering and science objectives
- A set of scoring functions to assess how well a sequence achieves the objectives

The DFX inputs are created using the modeling language of the Automated Planning/Scheduling Environment (ASPEN). ASPEN provides a user-friendly modeling syntax that defines the spacecraft in terms of activities, resources, states, and constraints. Once the spacecraft is fully specified, ASPEN uses its planning and scheduling engine to generate an operations sequence of events and scores the sequence in terms of science, operability, etc. The scoring criteria are completely defined by the user within the model. The result is that the design team gets quantitative feedback on how well the design achieves science objectives and meets operability constraints (such as meeting power margins, etc.). This type of tool is targeted at enabling design teams to rapidly and impartially evaluate large numbers of spacecraft designs with little effort, thus allowing improved analysis of design tradeoffs to enhance science and operations concerns for future missions. The DFX tool has been tailored to maximize the automation of the optimization process and minimize the amount of customization required by the user. Other work in using planning

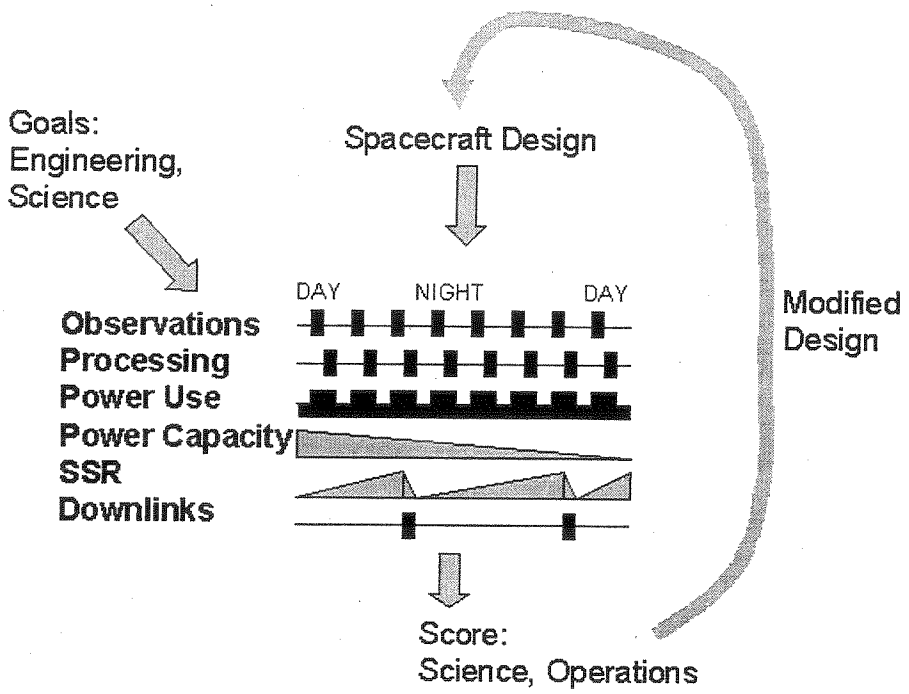


Figure 1: Intended Usage of DFX Tool

technology to evaluate mission requirements and mission designs includes [Ghallab, 1997].

The remainder of this paper is organized as follows. First, we briefly describe the basic planning and scheduling capability that we presume. We only outline this material because it has been covered in detail elsewhere. Then we describe a detailed application of the DFX concept to evaluation of competing designs for the Pluto Express Pre-project (now part of the Fire and Ice Program at the Jet Propulsion Laboratory). Next, we describe plans for the DFX project and conclusions.

2. Automated Planning and Scheduling

At the heart of DFX is an automated planning and scheduling capability. Planning is the selection and sequencing of activities such that they achieve one or more goals and satisfy a set of domain constraints. Scheduling selects among alternative plans and assigns resources and times for each activity so that the assignments obey the temporal restrictions between activities and the capacity limitations of a set of shared resources. In addition, scheduling is an optimization task in which metrics such as tardiness and make-span are minimized. Scheduling is a classical combinatorial problem that has long been studied by researchers in operations research.

Our DFX concept is built upon the ASPEN planning/scheduling system [Fukunaga et al. 1997]. In order to enable the rapid development of automated scheduling systems for NASA applications, we have developed ASPEN, a reusable, configurable, generic planning/scheduling application framework. ASPEN is an object-oriented system (implemented in C++) that provides a reusable set of software components that implement the elements commonly found in complex planning/scheduling systems.

ASPEN provides a basic planning and scheduling capability for DFX. This basic capability takes several inputs:

1. A model of the spacecraft, operations constraints and rules (including for science experiments), and standard operating procedures
2. A specification of a problem, consisting of an initial spacecraft state, exogenous events, and desired goals (science and engineering).

Using these inputs, an automated planning and scheduling system generates plans/schedules which achieve the goals (if possible) while obeying the relevant operations constraints.

This paper describes an initial demonstration performed using the DCAPS scheduler [Rabideau et al. 1997], but current efforts to continue the DFX concept are based on the ASPEN scheduler [Fukunaga et. al 1997] However, the DFX concept is general, so any general purpose automated planner/scheduler would be usable within the DFX concept. However, as the central idea behind the concept is to allow experimentation with a wide range of spacecraft designs and mission scenarios, if the spacecraft models and mission scenarios are easily changeable and modular, this increases the usability of the DFX tool.

3. Demonstration on Pluto Express Pointing Alternative Designs

The first proof-of-concept design of the DFX tool is based on the Pluto Express (PX) sciencecraft. PX is a robotic reconnaissance mission to Pluto and its moon Charon. The Pluto mission will be unique in its approach. In order to minimize cost, while containing the risks associated with lower cost, the Pluto mission is being conceived as a pair of very small spacecraft, using, where possible, lightweight advanced-technology hardware components and advanced software technology. The Pluto mission plan calls for launch of two spacecraft early in the next decade toward encounters with Pluto and Charon around 2010 or later. The science goals of PX are to: characterize the global geology and geomorphology of Pluto and Charon, imaging both sides of each; map the surface composition; and characterize Pluto's neutral atmosphere, including composition, thermal structure, and aerosol particles.

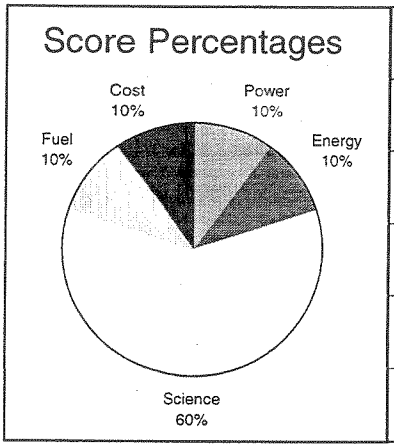
Pluto Express Model

The PX model is based on the assumption that the planetary and satellite encounter phase will drive the design. Because of the fast flyby velocity (12-20 km/s), the majority of science is performed within ±1 hour of encounter. Obviously, it is very important to optimize the science during these two hours. The preliminary model that we have built does not include the launch and cruise phases of the mission. Although there are two spacecraft, it is assumed their science will be identical. In reality, the two spacecraft encounters will be separated by 6 months and the science of the first will drive the science of the second. Indeed the science will be of the same type and with the same constraints so the assumption of identical science is valid. Another assumption in the model is that the start time and duration of both the Drop Zond (Charon probe) uplink and the spacecraft turns are fixed. This is a valid assumption because these times will not change between design options in the preliminary model.

The preliminary model consists of a common set of resources and a series of design options. Table 1 lists the common resources across all designs.

Quantity	Capacity	Resource	Power (W)	Mass (kg)
2	82 M RAM	local memory buffer		0.40
2	1 G RAM	mass memory buffer	2.15	0.40
2	n/a	general purpose heat sources	1.0	0.50
2	n/a	flight computers	5.22	0.40
1	n/a	integrated camera (UV, IR, visible)	6.0	6.90
1	n/a	hydrazine fuel tank	-	20.64
1	500 units	hydrazine fuel	-	16.5
6	n/a	delta-V thrusters	6.0	6.0
24	n/a	RCS thrusters	0.80	0.29
2	n/a	valve drive electronics (VDE) units	2.5	1.2
2	n/a	inertial reference units	4.00	0.40
2	n/a	stellar compass	0.50	1.5
1	n/a	high gain antenna	-	3.0
2	n/a	low gain antenna	-	1.0
2	n/a	telecommunications electronics	27.0	3.80

Table 1: Model Resources



Resource	How it is Scored	Effect on Score
Power	$\frac{\text{Max. Power Available - Max. Power used}}{\text{Max. Power Available}}$	more margin = higher score
Energy	$\frac{\text{Total Energy Avail. - Total Energy used}}{\text{Total Energy Available}}$	more margin = higher score
Science	$\frac{\text{Science data captured}}{\text{allocated memory}}$	more science = higher score
Fuel	$\frac{\text{Fuel remaining after encounter}}{\text{Total Fuel Available}}$	more margin = higher score
Cost	Each resource cost 1 $ unit	scan platform is 1 unit

Table 2: Score Resource Computations

The primary design option of the preliminary model is the method of science pointing. The camera will be mounted on the body of the spacecraft and pointed with either the spacecraft control system (e.g., thrusters) or a movable mirror (called a scan platform). Each spacecraft slew between science frames requires 9 seconds that cannot be used for science data collection. The movable mirror only takes 1 second to change the science pointing. Due to the short encounter duration, the longer slew time for the spacecraft thruster based pointing greatly reduces the overall science return. In addition, the movable mirror option does not require the cold gas thruster system used for fine pointing slews. The absence of the cold gas thruster tank, plumbing, and thrusters reduces the overall mass of the spacecraft. The scan platform option adds additional complexities because it has moving parts.

The output of the DFX tool is a science data acquisition plan (SDAP). The SDAP is scored based on resource utilization and science output. An overall score is computed based on weighted contributions of each of the resources. For the PX mission, a preliminary scoring strategy was based on the resources of fuel, cost, power and energy each contributing 10% to the overall score, and science return contributing the remaining 60%. The computations of these resources and their influence on the score are summarized in Table 2. For the preliminary model, this strategy was arbitrarily chosen. Normally, the science community and the design engineers would develop a scoring strategy applicable to the mission.

Although the cost capability is built in to the scoring strategy, there is very limited information regarding the cost of the components. For this reason, the cost does not affect the score significantly. Likewise, the preliminary model for propulsion does not have thruster efficiency and propellant usage. The model currently assumes a fixed amount of propellant usage for each spacecraft turn.

Table 3 shows the direct comparison of the design options modeled in the demonstration, with the key pointing model difference highlighted in grey. Figure 2 shows the trace of the DFX tool output, which shows the score computation in detail.

Design Element	Design One	Design Two
Pluto Express Option	1D (RPS, solid upper rocket stage)	1D (RPS, solid upper rocket stage)
RPS	Amtec	Amtec
Drop Zond	included	included
Pointing mechanism	scan platform	cold gas thrusters
IRU	New Millennium development	New Millennium development
VDE	New Millennium development	New Millennium development
Stellar Compass	New Millennium development	New Millennium development

Table 3: Design Options Demonstrated

4. Future Plans

We are currently working on adding more detailed design information and design options to the model. Table 4 lists several design options that the PX project is currently pursuing. We are also planning to add the launch and cruise phases to the model. We would like to get good cost data for each of the spacecraft components and operations costs built into the model. We will be adding a

Options 1:	Option 2:
• RPS, solid upper rocket stage • The RPS is an Amtec • There is a Drop Zond included • There is a scanning platform • New Millennium development on IRU, VDE, and Stellar Compass	• RPS, solid upper rocket stage • The RPS is an Amtec • There is a Drop Zond included • There is no scanning platform • New Millennium development on IRU, VDE, and Stellar Compass
Scoring Results	Scoring Results
Allocated power: 85 Allocated energy: 627300 Allocated fuel: 500.0 Allocated memory: 200.0 Allocated cost: 100 Max. power used: 8 Total power switches: 46.0 Total energy used: 13096.0 Total fuel left: 498.0 Total data stored: 149.47 Total data taken: 149.47 Total observations: 48 Total cost: 36.0	Allocated power: 85 Allocated energy: 627300 Allocated fuel: 550 Allocated memory: 200.0 Allocated cost: 100 Max. power used: 8 Total power switches: 23.0 Total energy used: 13708.80 Total fuel left: 546.80 Total data stored: 135.62 Total data taken: 135.62 Total observations: 20 Total cost: 37.0
Power score: 90.6 Energy score: 97.9 Fuel score: 99.6 Cost score: 64.0 Data score: 74.7	Power score: 90.6 Energy score: 97.8 Fuel score: 99.4 Cost score: 63.0 Data score: 67.8
Total Score: 80.05 / 100	Total Score: 75.77 / 100

Figure 2: Trace of Output for Scoring the Two Alternative Pluto Express Pointing Designs

Trajectory	Telecommunications	Power
• Direct trajectory • JGA with solid • rocket motor – VVJGA – VVVJGA - JGA with solar • electric propulsion • and solar array	HGA: • Carbon-Carbon • Composite MGA: • Articulated • Passive Downlink: • Ka band • X band • both • optical comm. experiment	• 2 brick RPS • 3 brick RPS • solar array for solar electric propulsion

Table 4: Future Design Options

variable parameter space and running several scenarios over the space to identify candidate designs which optimize both science and operability. As mentioned earlier, we are going to work with PX engineers to determine a better scoring strategy.

5. Conclusions

This paper has described the novel application of using automated planning and scheduling technology to provide more concrete evaluations of alternative spacecraft designs and mission scenarios. In this application, called DFX, the planning/scheduling engine is used to produce baseline sequences for various combinations of spacecraft designs and/or mission scenarios. These sequences are then evaluated with respect to end mission success criteria such as science return, resource margins, and other metrics. This rapid evaluation is enabled by the automated nature of planning/scheduling technology and relieves the spacecraft and mission design personnel from the burden of manually constructing strawman sequences for evaluation. This enables the evaluation of a wider range of designs for both the spacecraft and overall mission. In that the spacecraft and mission specification can be made modular, construction of a wide range of spacecraft and mission designs can be facilitated. The DFX concept has been demonstrated using the DCAPS planning/scheduling engine on two pointing alternatives for the Pluto Express mission.

References/Sources

[Eggemeyer, 1995] W. Eggemeyer, "Plan-It2 Bible," JPL Technical Document, 1995.

[Fukunaga et al, 1997] A. Fukunaga, G. Rabideau, S. Chien, and D. Yan, "ASPEN: A Framework for Automated Planning and Scheduling of Spacecraft Control and Operations," Proceedings of the International Symposium on Artificial Intelligence, Robotics, and Automation in Space (I-SAIRAS97), Tokyo, Japan, 1997.

[Ghallab 1997] M. Ghallab, LAAS/CNRS, Toulouse, France, Personal Communication, September 1997.

[Pluto 1996a] Pluto Express FY 96 Annual Report, JPL Technical Document, 1996.

[Pluto 1995] Pluto Express FY 95 Annual Report, 1995

[Pluto 1996b] Pluto Express Power and Mass Worksheet, 1996.

[Pluto 1996c] Pluto Information Express Home Page, 1996.

[Price et al. 1996] H.W. Price, J.B. Carraway, S.E. Matousek, R.L. Staehle, R.J. Terrile, E.J. Wyatt, Pluto Express Sciencecraft System Design, JPL Technical Document, 1996.

[Rabideau et al. 1997] G. Rabideau, S. Chien, T. Mann. C. Eggemeyer, J. Wilis, S. Siewert, P. Stone, "Interactive Repair-based Planning and Scheduling for Shuttle Payload Operations," Proceedings of the IEEE Aerospace Conference, Aspen, CO, 1997.

Extended Abstract:
TOWARDS SELF-RELIANT AUTONOMOUS SYSTEMS

Reid Simmons, Sven Koenig, Joaquin Lopez
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Richard Goodwin
IBM Watson Research

To achieve long-term autonomy, autonomous systems must be extremely reliable. We are investigating a number of different techniques to achieve reliability, specifically including the use of probabilistic representations, decision-theoretic planning, and the use of execution monitoring and error recovery strategies that handle situations when plans fail to meet their expectations.

One hallmark of autonomy is the ability to react appropriately to a variety of situations. This presumes that the agent can perceive what situation it is in, and knows the appropriate reactions for that situation. The reason this is difficult for systems that interact with the real world is due to uncertainty, which can arise from uncertainty about the effects of actions, uncertainty in perceiving the environment, and uncertainty in one's model of the environment. One way to handle this is to maintain explicit models of the agent's uncertainty and track when it is outside of the expected bounds of its current plan; Another (orthogonal, but not mutually exclusive) approach is to generate plans that reduce potential uncertainty (e.g., use actions with more certain outcomes, or make sure the agent stays in areas where it has more certain knowledge of the environment).

We have explored both these methods, mainly in the context of mobile robots. In the area of maintaining and tracking explicit models of uncertainty, we make use of Partially Observable Markov Decision Process (POMDP) models to model all three sources of uncertainty [Simmons 95]. POMDP models are automatically compiled from topological maps of the environment, augmented with approximate metric information. As the robot moves, the probability distribution of the POMDP model is updated, using Bayes' rule, to reflect the probable change in robot position. As the robot senses its environment, the observations are used to further update the probability distributions. In this way, while the robot is never exactly sure of its current state, neither does it ever (or rarely) get completely lost. This technique has enabled Xavier, an indoor delivery robot [Simmons 97], to navigate in a natural, peopled environment with greater than 95% success rate in achieving its high-level navigation goals. In more space-relevant applications, we could easily apply these same, or similar, techniques to long-distance navigation of planetary rovers [Krotkov 95] and to tracking the state of autonomous spacecraft (similar to the approach taken by [Williams 96]).

As mentioned, an orthogonal, but compatible, approach is to reduce the potential for uncertainty by judicious choice of plans. This is especially important in situations that are time critical, where replanning or recovering from unexpected situations may not be feasible. Our approach here is to use decision-theoretic planning techniques to create plans that maximize the expected utility of the agent, where utility indicates the agent's preferences (including its risk attitude) [Koenig 96]. The problem here is that the space of such plans is very large. To make planning tractable, the agent needs to focus planning effort on plans that are likely to be of high utility. In particular, we need to determine which plans to focus on, how to refine abstract plans, and when to stop planning and begin execution. This is done by maintaining upper and lower bounds on expected utility, and using a form of sensitivity analysis to rank potential plans and prune out those plans that are guaranteed not to be optimal [Goodwin 95]. This planning approach has been successfully applied to

Xavier, and is also highly applicable to autonomous rovers and spacecraft, especially in dealing with resource-limited situations (such as encounters).

Finally, we have developed an overall architectural framework, based on the Task Control Architecture (TCA), to integrate monitoring and exception handling with task planning and execution. The basic principle, called *structured control*, [Simmons 94a] is that a useful way of developing reliable autonomous systems is to start with behaviors that achieve goals in nominal situations, and then incrementally layer on reactive behaviors (monitors and exception handlers) to handle exceptions. TCA supports the ability to incrementally add monitors and exception handlers to existing systems, making it possible to add robustness to a system without having to change its existing components. This methodology has been applied to several mobile robots [Simmons 92, Simmons 94b], and has demonstrated significant improvements in their abilities to operate reliably for long periods of time.

More recently, we have been investigating how to add additional structure to the monitors and exception handlers to increase their coverage. The idea is to structure the monitors hierarchically, so that high-level monitors are used to detect general categories of exceptional situations (and have associated high-level methods for handling the exceptions), while lower-level monitors are tuned to detect, and handle, more specific situations. For example, in navigating to a given destination, a high-level monitor may checks whether the robot is failing to make progress towards its goal -- if that monitor triggers, the response is to plan an alternate route, or to inform a human if no alternative exists. Lower-level monitors, on the other hand, check for more specific situations (e.g., is the robot trapped in a cul-de-sac? is it getting low on battery voltage?), and likewise trigger more specific recovery strategies (e.g., back up, recharge).

The probabilistic and execution monitoring approaches interact in that the monitors need to know the (likely) state of the world, and need to know what is considered to be "exceptional". The POMDP models can be used for tracking (likely) state; the probabilistic models used by the planner can be used to generate expectations on what should be happening, such as how long the rover should be expected to travel between two points. Integrating this probabilistic information with the monitoring and exception handling approach is still ongoing research. Additional ongoing research addresses the problem of generating such monitors automatically from models of the agent and its environment. This will likely incorporate both probabilistic and symbolic (model-based) reasoning techniques [Williams 96].

In conclusion, autonomous agents cannot blindly execute their plans -- they need extensive capabilities to monitor the state of the world robustly and react appropriately. We are researching several techniques to address this problem, incorporating work in probabilistic reasoning, decision-theoretic planning, architectures for autonomous systems, and model-based reasoning.

References

[Goodwin 95] R. Goodwin. "Sensitivity Analysis for Meta-level Planning", *International Joint Conference on Artificial Intelligence*, Montreal Canada, July 1995.

[Koenig 96] S. Koenig, R. Goodwin and R. Simmons. "Robot Navigation with Markov Models: A Framework for Path Planning and Learning with Limited Computational Resources", Lecture Notes in Artificial Intelligence, *Volume 1093: Reasoning with Uncertainty in Robotics*; L. Dorst, M. van Lambalgen, and R. Voorbraak (Eds.), Springer, pp. 322-337, 1996.

[Krotkov 95] E. Krotkov M. Hebert and R. Simmons. "Stereo Perception and Dead Reckoning for a Prototype Lunar Rover", *Autonomous Robots*, **2:4**, pp. 313-331, 1995.

[Simmons 92] R. Simmons. "Monitoring and Error Recovery for Autonomous Walking" *IEEE International Workshop on Intelligent Robots and Systems*, pp. 1407-1412, July, 1992.

[Simmons 94a] R. Simmons. "Structured Control for Autonomous Robots", *IEEE Transactions on Robotics and Automation*, **10:1**, pp. 34-43, February 1994.

[Simmons 94b] R. Simmons. "Becoming Increasingly Reliable", *Second International Conference on Artificial Intelligence Planning Systems*, Chicago, IL, June 1994.

[Simmons 95] R. Simmons and S. Koenig. "Probabilistic Robot Navigation in Partially Observable Environments", *International Joint Conference on Artificial Intelligence*, Montreal Canada, August 1995.

[Simmons 97] R. Simmons, R. Goodwin, K. Zita Haigh, S. Koenig and J. O'Sullivan. "A Layered Architecture for Office Delivery Robots", *First International Conference on Autonomous Agents*, Marina del Rey, CA, February 1997.

[Williams 96] B. Williams and P. Nayak. "Immobile Robots: AI in the New Millennium", *AI Magazine*, Fall 1996.

Knowledge Acquisition for the Onboard Planner of an Autonomous Spacecraft

Benjamin D. Smith
Jet Propulsion Laboratory
4800 Oak Grove Drive M/S 525-3660
Pasadena, CA 91109-8099
smith@aig.jpl.nasa.gov

Kanna Rajan and **Nicola Muscettola**
NASA Ames Research Center
Mail Stop 269-2
Moffett Field, CA 94035
{kanna,mus}@ptolemy.arc.nasa.gov

Abstract

Deep Space One (DS1) will be the first spacecraft to be controlled by an autonomous closed loop system potentially capable of carrying out a complete mission with minimal commanding from Earth. A major component of the autonomous flight software is an onboard planner/scheduler. Based on generative planning and temporal reasoning technologies, the planner/scheduler transforms abstract goals into detailed tasks to be executed within resource and time limits. This paper discusses the knowledge acquisition issues involved in transitioning this novel technology into spacecraft flight software, developing the planner in the context of a large software project and completing the work under a compressed development schedule. Our experience shows that the planning framework used is adequate to address the challenges of DS1 and future autonomous spacecraft systems, and it points to a series of open technological challenges in developing methodologies and tools for knowledge acquisition and validation.

Introduction

The future of the space program calls for ambitious missions of exploration and scientific discovery. Searching for life on Mars, Europa and elsewhere in the solar system and beyond will require the solution of several challenging technical and organizational problems. A central one is the implementation of increasingly capable and autonomous control systems to ensure both mission accomplishment and mission safety (Williams and Nayak Fall 1996; Hayes-Roth 1995; Tambe *et al.* 1995). Without these systems missions will have to be run with the current, traditional approach. This relies on frequent communication with Earth and teams of human experts guiding step by step a mission through its tasks and analyzing and reacting to the occurrence of malfunctions. The cost and logistics difficulties of this approach, however, are so high that it cannot be reasonably carried over to the expected growth of missions and mission capabilities. Autonomy technology is an answer to these problems.

The Remote Agent (RA) (Pell *et al.* 1996a; 1997) will be the first artificial intelligence-based autonomy architecture to reside in the flight processor of a spacecraft and control it for 6 days without ground intervention. The mission on which RA will fly is Deep Space One (DS1), the first deep-space mission of NASA's New Millennium Project. RA achieves its high level of autonomy by using an architecture with three components: an integrated planning and scheduling system (PS) that generates sequences of actions (plans) from high-level goals, a intelligent executive (EXEC) that carries out those actions and can respond to execution time anomalies, and a Model-based Identification and Recovery system (MIR) that identifies faults and suggests repair strategies. Each module covers a different function in the architecture and uses a different computational approach. One characteristic however is common to all of them: the reliance on models of the domain that are largely independent from the task to be fulfilled. These models allow the module to rely on a much deeper understanding of the structural characteristics of the domain than possible with classical rule-based approaches, facilitating model analysis and model reuse.

This paper discusses the knowledge acquisition process used for models and heuristics of the planning and scheduling system (PS) of DS1. We started the process with an approach to planning knowledge representation (Muscettola 1994) that had been demonstrated in a rapid-prototype effort (Pell *et al.* 1996a). With DS1 we had to face additional challenges due to having to develop PS in the context of the development of the full flight software, to the additional complexity of the domain, to the compressed schedule for development and to the risk-management requirements. Also, architectural solutions internal to RA had to be enhanced due both to the increase in capabilities that were needed to control a real spacecraft and to the need to provide sounder software engineering approaches. We will describe how the knowledge acquisition process was carried out and the strengths and weaknesses we found in our current approach.

Section deals with the Remote Agent software architecture highlighting the details of the planner and it's plan representation. Section deals with issues in

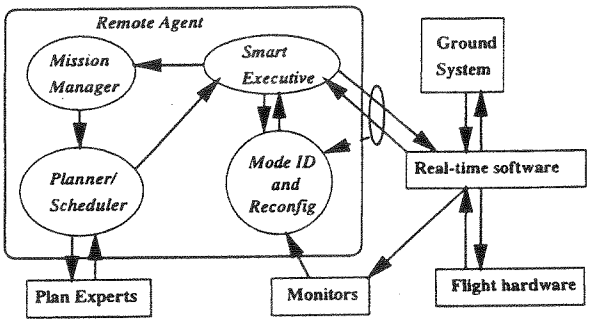


Figure 1: RA architecture

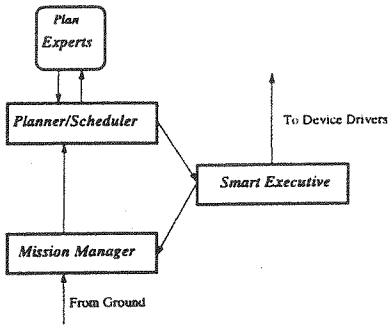


Figure 2: Planner/Scheduler Architecture

knowledge acquisition including references to the spiral development process, model acquisition and interfaces to external experts. Section deals with the open issues as a result of the development process including the need for validation and debugging tools. Conclusions appear in Section .

The Remote Agent and Planner Architectures

The RA architecture consists of four distinct components (Figure 1), the *Planner/Scheduler*, the *Mission Manager* (Muscettola *et al.* 1997), the *Smart Executive* (EXEC) (Pell *et al.* 1996b) and the *Mode Identification and Recovery* (MIR) system (Williams and Nayak 1996; Fall 1996) .

The execution of plans by the EXEC is achieved by interaction with a Mode Identification system and a lower level real time monitoring and control component. MIR provides the EXEC with a level of abstraction to reason about the state of the various devices it commands. The *monitoring* layer takes the raw sensor data and discretizes it to the level of abstraction needed by MIR. Finally, the *control and real-time system* layer takes commands from the executive and provides the actual control of the low level state of the spacecraft. It is responsible for providing the low level sensor data stream to the monitors. Details of the Remote Agent architecture can be found in (Pell *et al.* 1996a; 1997).

The planner/scheduler (PS) generates a detailed plan of action from a handful of high-level goals, based on knowledge of the spacecraft contained in a domain model. The model describes the set of actions, how goals decompose into actions, the constraints among actions, and resource utilization by the actions. The planning engine searches the space of possible plans for one that satisfies the constraints and achieves the goals. The action definitions determine the space of plans. The constraints determine which of these plans are legal, and heavily prune the search space. The heuristics guide the search in order to increase the number of plans that can be found within the time allocated for planning.

Figure 2 describes the overall view of the PS. The Mission Manager (MM) contains the long term mission plan with goals for the entire mission. Ground operators can interact with the MM to add, remove or edit goals in the mission plan. When the EXEC requests a new plan from the MM, the MM selects a new set of goals from the mission profile and combines them with the initial state provided by the EXEC and generates a partial plan for the planner. When the EXEC has almost finished executing the plan, it requests a new plan from the MM and appends it to the current one. For the RA experiment the plan horizon will consist of two three-day segments.

Knowledge Representation of the Planner

The knowledge representation of the planner is distributed among the domain models, the planner heuristics, the mission profile and the plan experts. The domain models encode the behavioral and operational constraints imposed on the spacecraft by the mission and the hardware. The heuristics guide the planner search to decrease the computational resources needed to find a plan and to increase plan quality. The mission profile encodes the long term goals and mission requirements as determined by the ground controllers and mission designers, and resides in the Mission Manager's temporal database. Finally the plan experts are special-purpose software modules, written and maintained by other teams, with which the planner interacts to obtain knowledge that cannot be easily encoded in the plan model.

Model Representation. The PS uses a hybrid planning/scheduling representation that models continuous processes on parallel timelines to describe actions, states and resource allocations. PS provides also for temporal and parametric flexibility and uses planning experts.

Plans consist of several parallel *timelines*, each of which consists of a sequence of *tokens*. A timeline describes the evolution of a spacecraft state over time, and the tokens describe those states. For example, consider one timeline that describes the main engine.

```
(Define_Compatibility
  ;; compats on SEP_Thrusting
  (SEP_Thrusting ?heading ?level ?duration)
  :compatibility_spec
  (AND (equal (DELTA MULTIPLE
                (Power) (+ 2416 Used)))
       (contained_by
         (Constant_Pointing ?heading))
       (met_by (SEP_Standby))
       (meets  (SEP_Standby))))

(Define_Compatibility
  ;; Transitional Pointing
  (Transitional_Pointing ?from ?to ?legal)
  :parameter_functions
    (?_duration_ <- APE_Slew_Duration
        (?from ?to ?_start_time_))
    (?_legal_    <-
      APE_Slew_Legality
        (?from ?to ?_start_time_))
  :compatibility_spec
  (AND (met_by (Constant_Pointing ?from))
       (meets  (Constant_Pointing ?to))))

(Define_Compatibility
  ;; Constant Pointing
  (Constant_Pointing ?target)
  :compatibility_spec
  (AND (met_by (Transitional_Pointing
                 * ?target LEGAL))
       (meets  (Constant_Pointing
                  ?target * LEGAL))))
```

Figure 3: An example of a compatibility constraint in the planner model.

If the plan is to start in standby, fire up the engine, and return to standby, the timeline would have one token for each of those processes. Each token has a start time, and end time, and a duration. Each token can have zero or more arguments (e.g., the thrust level at which to fire the engine).

The plan model consists of definitions for all the timelines, definitions for all the tokens that can appear on those timelines, and a set of temporal constraints that must hold among the tokens in a valid plan. The planner model is described in a domain description language (DDL), and is represented as part of the planner's data base also called the Plan DB.

Temporal constraints are specified in DDL by *compatibilities*. A compatibility consists of a *master token* and a boolean expression of temporal relations that must hold between the master token and *target tokens*. An example is shown in Figure 3.

The first compatibility says that the master token, SEP_THRUSTING (when the Solar Electric Propulsion engine is producing thrust), must be immediately preceded and followed by a standby token, temporally contained by a constant pointing token, and requires 2416 Watts of power. Constant pointing implies that the

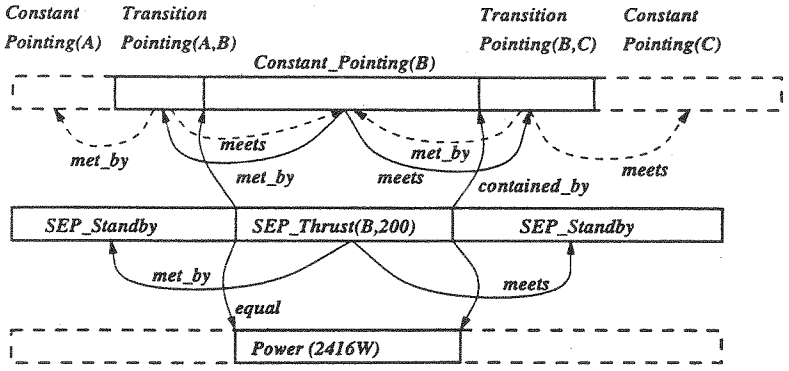


Figure 4: Plan Fragment

spacecraft is in a steady state aiming its camera towards a fixed target in space. Transitional pointings turn the spacecraft. The SEP standby state indicates that the engine is not thrusting but has not been completely shut off. A plan fragment based on these compatibilities is shown in Figure 4.

Heuristics. Heuristics guide every choice point of the planners search. On each iteration of the search, the planner chooses an unresolved compatibility constraint and a way to resolve it: by constraining an existing token to satisfy the constraint, adding a new token that satisfies it, or assuming that it will be satisfied by some token in the next horizon. There are other decisions as well, such as grounding under-constrained argument values. For all of these decision points a heuristic can be provided that tells the planner in what order to try the alternatives and which alternatives should never be considered.

Mission Profile. The goals for the entire mission are stored in an on-board file called the *mission profile*, which is managed by the Mission Manager. The profile captures mission operations knowledge, such as when the communications passes are scheduled, how much fuel is allocated for each segment of the mission, when various mission phases start and stop, and so on. The profile also serves as the primary interface with the ground controllers. The ground team commands the spacecraft at a high level by changing or adding goals to the profile.

Plan Experts. A large software project like the DS1, requires the contribution of several teams with specialized knowledge. *Planning Experts* are programs developed and maintained by other teams. They coordinate with the planner but which are not strictly part of its domain representation.

A prime example is the Attitude control Planning Expert (APE), which answers queries about the duration of a turn and whether a turn violates pointing constraints (e.g., will it expose the camera to a damaging bright radiation source). How violation con-

straints are calculated is completely opaque to the planner. As a result, separating the plan experts from the planning model simplifies the knowledge acquisition and software maintenance process. Quite often, due to the specificity of these modules, the code is also reusable across missions. For instance, much of the code for attitude constraint violation in APE came from NASA/ESA's Cassini mission (G.M.Brown *et al.* 1995).

There are two kinds of plan experts. The first kind answers questions about constraints. APE is of this variety. The second kind generates goals for the planner to achieve. These on-board goal generators allow the spacecraft to make autonomous decisions, within certain parameters, based on local information. The prime example on DS1 is the on-board navigator, which provides goals on trajectory related maneuvers and goals for images of nearby celestial bodies from which NAV can determine the spacecraft position.

The planner asks the goal generators for goals when the planner is ready for them. The goal generators have no visibility into the plan, other than whatever information provided in the request. When the goals are returned, the planner decides how they will be achieved in the plan, or whether they are achieved at all. If the plan is over-constrained, goals can be rejected based on a global prioritization scheme.

The Knowledge Acquisition Process

Traditionally flight software for a spacecraft consists only of low level device drivers, attitude control system and simple sequence execution capabilities. Commanding done from ground allows the operational and mission constraints to be designed and implemented at a later time, sometimes even after launch. With on-board autonomy, the design process must take a more comprehensive view to the full mission life cycle including from the *very beginning* the modes, operations and expected behaviors of the spacecraft in the domain models. To accomplish this we used a spiral development model (Boehm 1988).

In the following sections we discuss the knowledge acquisition process and methodology for the planner and the resulting problems and issues they raised.

The Spiral Development Process

In spiral development (Boehm 1988), functionality is added incrementally in distinct software releases. This allows base functionality to be understood and developed before moving to more complex functionality. Processes and standards are also refined in each spiral. At the end of each development cycle, project teams meet to discuss the obstacles they encountered and the lessons they learned. The DS1 spiral process is discussed further in (Krasner and Bernard 1997).

At the beginning of each spiral, the mission engineers created a baseline scenario that would exercise the new functionality for that spiral while still requiring the old functionality. The hardware management team (HWMT) then arranged several days of knowledge acquisition meetings with the hardware developers, who would detail the software requirements for their hardware to work correctly.

Subsystem	R1	R2	R3
Mission events	0	1	3
Power	0	0	2
Ion Propulsion	1	5	5
Attitude control	3	4	4
Communications	0	1	2
MICAS	1	1	6
Beacon experiment	0	0	2
RCS system	1	1	3
Navigation	3	3	4
Planner/scheduler	1	1	1
Total	10	17	32

Table 1: Number of timelines changed in the model for each development release.

Each of the modeling and software development teams sent representatives to these meetings. The hardware developers presented the baseline behavior for the upcoming spiral, and the modelers asked questions to elicit further details. Since each component of the RA models the hardware at a different level, having representatives from each team was particularly helpful in identifying interaction issues across the different levels.

The DS1 Spiral releases were designated R1 through R3. To give the reader the scope of development that took place, we show the evolution of the planner model sizes for each revision in Tables 1, 2, and 3.

From the PS perspective each revision in the spiral development model involved successively sophisticated constraint modeling of the spacecraft. In the first revision the model only dealt with simple turns and picture taking for navigation images; more complex issues such as power, thermal modeling were ignored. In the next revision the model included the modeling the SEP engines and obtaining more detailed trajectory information from the navigation expert. The third spiral release added power management, advanced turns, and comet fly-by related activities.

In each revision of the Spiral development approximately eight weeks were needed for knowledge capture and another eight weeks for model development and tuning of the planner search.

Model Acquisition

Model acquisition in each cycle started with the cognizant system engineer specifying the baseline functionality to be covered, layered on top of designs of subsystems already implemented. Each team then devel-

Subsystem	R1	R2				R3			
	Tot	Add	Mod	Del	Tot	Add	Mod	Del	Tot
Mission events	0	1			1	5			6
Power	0				0	3			3
Ion Propulsion	1	11	1		12	1	3		13
Attitude control	4	4			8	6	2	1	14
Communications	0	3			3	2	2		5
MICAS	3	5			8	14			22
Beacon experiment	0				0	4			4
RCS system	1				1	4	1		5
Navigation	6		2		6	3			9
Planner/scheduler	2				2				2
Total	17				41				69

Table 2: Number of token modifications to the model for each development release.

Subsystem	R1	R2	R3
Mission events	0	2	4
Power	0	0	0
Ion Propulsion	3	11	14
Attitude control	2	11	16
Communications	0	3	7
MICAS	3	3	20
Beacon experiment	0	0	4
RCS system	0	0	2
Navigation	4	3	7
Planner/scheduler	2	2	2
Total	16	36	76

Table 3: Number of compatibilities changed in the model for each development release.

oped a specification called a *Problem Statement* which described how that functionality would be achieved. For the planner, this described changes to the planning model and engine and described changes to interfaces with the EXEC and plan experts. Teams with interfaces with the planner (especially the EXEC) would comment and propose design changes and any additional requirements. After a few iterations of this process, the modeler would update the *Token Dictionary*. The token dictionary details the syntax and semantics of each token type on all the timelines and forms the primary document for all negotiated informal interfaces with the EXEC.

We used an informal elicitation methodology to acquire the models described in the problem statements. The elicitation began with a standard list of questions about how the subsystem operated and what constraints or interactions it had with other subsystems. These would then lead to more detailed questions. The captured knowledge was compiled in a semi-formal document and approved by the engineers in a separate session. There was no formal methodology to ensure coverage other than the constraints of the baseline scenario. Nonetheless, this was adequate to build the plan models and successfully complete the scenarios for each spiral release. In retrospect, an acquisition methodology that resulted in a formal specification with guaranteed coverage would have been useful for rigorously validating the model (see Section).

Issues in Domain Modeling. In modeling for the DS1 mission, we discovered that a relatively large number of modeling tasks were easy to do, given the syntax and semantics of DDL. In a couple of cases we had to introduce auxiliary timelines to support the planner's reasoning process.

With each iteration of the development cycle the fidelity of the planner models was increased. Knowledge acquisition from each spiral cycle affected the planner's domain model and it's heuristics. Changes to the domain model itself were fairly straightforward. However, these changes often required re-tuning the heuristics which consumed significant development effort.

Heuristics. Because of the tight coupling of the domain model to the heuristics, changes to the model almost always require corresponding changes to the heuristics. This makes it difficult to introduce incremental changes to the model. Normally, a family of timelines corresponding to a new device or capability can be added with minimal impact on other timelines. Most of the constraints are among the timelines in the family, with a handful of constraints to external resources such as power or spacecraft attitude. However, the new timelines change the optimal search strategy, and this requires the heuristics to be re-tuned.

Unmodeled Activities

Sometimes the ground controllers want to execute unusual maneuvers that are not modeled the planner. For example, they may want to execute a high-speed turn (normally disallowed) in order to jar loose a stuck solar panel. The model must provide a way for the ground controllers to execute contingency maneuvers such as

this without uploading a new model. Changing the model maybe fine for permanent patches, but the time and cost needed to develop and test the patch makes them impractical for one-time emergency maneuvers.

In support of this requirement the model provides a `special activity` token that can stand in for any activity the ground wants to execute but is not otherwise supported by the planning model. The ground controllers insert the token where they want it in the mission profile. It can be scheduled for a specific time, or scheduled relative to other events. The activities performed by the token are specified in a file of time-tagged low-level commands that the EXEC executes at the beginning of the token.

Since the actions executed in the special activity token are not modeled by the planner, it is possible that they could conflict with planned activities. For example, the plan could be trying to hold the spacecraft still in order to take an image while the special activity token is executing a high speed turn. To avoid such conflicts, constraints can be specified between the special activity token and other tokens in the plan. In this case, attitude dependent activities would be disallowed while the special activity token was active. These constraints can be specified in the mission profile.

Interfaces

The interfaces between the planner and other parts of the flight software impact the knowledge acquisition and representation. The planner has two main interfaces: interfaces with *plan experts*, and interfaces with the *Smart Executive* component of the RA.

Plan Expert Interfaces. Negotiating the plan expert interfaces was among the easiest of the knowledge acquisition tasks. This is largely due to the opaqueness of the plan experts to the planner and vice versa. The bulk of the knowledge acquisition was in the very first meeting, where the focus was understanding how the plan expert worked and explaining planner concepts to the plan expert developers. In the case of APE, the planning team needed to understand how to specify a turn, and what information was needed for APE to compute a turn. The details of how turns are computed were irrelevant.

Once this initial knowledge acquisition was completed, subsequent interface negotiations were completed in a matter of hours, usually by phone or email. The interfaces were formally defined as C structures that specified the information passing from the planner to the plan expert and back. These were captured in an interface control document, and in an executable interface specification language.

In some cases, the planner used assumptions about the inner workings of the plan expert to improve efficiency. For example, the legality and duration of a turn changes slowly and continuously over time. This allows a turn to be moved a couple hours ahead or back in the plan as needed without affecting its duration or legality. The planner model and heuristics exploited this knowledge to simplify the design and speed up the search. Assumptions of this sort were rare, and captured explicitly in the interface control documents.

EXEC Interfaces. The interfaces between the planner and the Smart Executive (EXEC) are embodied by the timeline and token definitions included in the planner's model.

In order for the EXEC to execute plans, it must know what tokens can appear in the plan and how to expand those tokens into detailed commands to the real-time flight software. This creates a very strong coupling between EXEC and the planner. All of the timelines, tokens, and their semantics were negotiated at the beginning of each spiral before any implementation took place. However, if the need for another token was discovered during development, or some token needed another argument, or the semantics were wrong, then the EXEC and PS had to change their implementations accordingly. Because the tokens are such a major part of the model implementation, changes of this sort occurred in every development spiral despite strong efforts to prevent them.

Several solutions to this interface issue were considered for DS1. One successful approach was to use *information hiding* to create private token arguments. Additional arguments are often needed to hold values derived from other arguments, or to propagate values from other tokens. The need for these arguments often goes unnoticed until development begins in earnest. Private arguments are seen by the planner, but are do not appear in the plan. This allowed modelers to add arguments for bookkeeping and propagation purposes without impacting the EXEC. This capability was introduced at the end of the R2 spiral, and used with great success in R3.

Interface Management Process. To ensure disconnects were kept to a minimum, another requirement added by the project during the design phase of each revision, was the development of *Problem Statements*, with details of each modules' design, interfaces and assumptions for that revision cycle. The planner in addition also had a token dictionary with the negotiated token level interfaces with the EXEC. With the EXEC with which the planner representation was tightly coupled, any agreements and assumptions in the planner's model were accurately document and easily accessible via a world wide web (WWW) interface to the dictionary and disconnects caught early on. In order to avoid disconnects with respect to the hardware specifications, especially as hardware delivery quantified the performance, the HWMT was the central point of contact for disseminating information.

The project also made sure that after the interface parties and the design phase for each cycle, but *before* the various teams started actually developing code, a

concept review would take place. Each team would publish a short document detailing their design choices and the assumptions made, especially towards generating the scenario in the current cycle and the interface requirements. Any disconnects found would require the project to follow through with the team in question to ensure the new design actually covered the complete scenario.

Distributed Development

Because of geographically distributed teams, design documents and interface agreements were exchanged primarily via a WWW interface with auto-posting features as mentioned in (Compton *et al.* 1997). The use of an intranet was decisive in successfully collaborating among remote sites especially when exchanging device level knowledge.

Additionally, for short design and concept reviews we used an *arachno-conference* where several people engaged in a conference call while accessing the web to view documents. This greatly reduced the time, effort and expense of commuting to a central site. Note also that the source code was under revision control.

Open Issues

The DS1 project presented several challenges in knowledge acquisition, representation, and validation. The DS1 planner proved capable of addressing these issues, at least to the extent needed to satisfy the requirements of DS1. However, there are a number of issues that must be resolved before this technology can be used on a risk-intolerant science mission by spacecraft engineers with minimal support from the planner development team.

Acquiring Heuristics is Difficult

Good heuristics are needed to make the planner search computationally tractable. Heuristics tell the planner what decisions are most likely to be best at each choice point in the planner search algorithm, thereby reducing the search. Developing a good set of heuristics for the DS1 planner is currently very difficult, both because it requires an intimate knowledge of how the planner search algorithm interacts with the model, and because the planner requires exceptionally good heuristics to achieve computational tractability. The DS1 model developers had this experience and so were able to develop good heuristics, but these obstacles must be overcome before spacecraft engineers can be expected to develop heuristics on their own.

One solution is to provide tools that derive heuristics automatically. Such tools have been discussed in the machine learning and planning literature. Two of the more promising approaches are to derive heuristics automatically through a static analysis of the plan model (Etzioni 1993) or to learn them by watching the behavior of the planner over several runs on a given model (Minton 1996). Unfortunately, the DS1 planner requires exceptionally good heuristics to achieve tractability, and these methods generally do not produce heuristics of that caliber. The sensitivity of the planner to the heuristic must be reduced before automatic heuristic acquisition can be feasible.

Development and Debugging Tools Needed

Modeling could be made considerably easier with even a few simple tools. Although there was insufficient time to develop them for DS1, our experience with developing and debugging models suggested a number of desirable features. Developing tools along these lines is one of our near term goals.

Plan Visualization Tools. One problem with the current system is that it is very difficult to understand what the planner is doing, despite copious output. This makes it difficult to isolate the decisions that lead to bugs in the plan, or prevent the planner from finding any plan at all. A visualization tool would help modelers to track the planners behavior, as well as making it easier for new users to understand how the planner works.

Deactivating Timelines. When debugging, the modeler often suspects the bug is within a small family of timelines. But as the model gets more complex, it becomes difficult to focus on the behavior of those timelines. A simple way to address this problem is to disable irrelevant timelines. The planner ignores the timelines and all compatibilities associated with them. This facility is rather easy to add, though there was insufficient time to implement given the compressed DS1 schedule.

Model Visualization Tools. As the model gets larger, it becomes harder to keep in mind all the constraints among the parts of the model. A model visualization tool that displayed a graphic view of the model (or a subset) and the constraints would help the modeler view this information as a whole.

Validation of the Planner

Before any spacecraft is launched, its flight software must be throughly tested and validated. The same is true for autonomous flight software. However, the validation methods used for traditional software are not generally applicable to autonomous software. New methods must be developed.

The planner can generate thousands of plans, depending on the mission goals, the spacecraft state, goals generated on-board by plan experts and variations on the model parameters. To fully validate the planner, one must be sure that it will generate a correct plan for every one of those possible situations, and that the plan can be executed correctly by the EXEC.

Tools for automatically generating and validating plans can greatly reduce this cost. One of the tools

we considered for DS1 but did not have time to implement was a constraint checker that tested whether the plan satisfied certain correctness constraints. These include the constraints in the model, plus additional constraints derived from flight rules and other operational constraints.

An alternative to generating and testing plans is to validate the model and verify that the planner always produces plans that are consistent with the model. One approach is to capture the flight rules and other requirements formally as constraints, and ensure that the model is consistent with all of them. A related possibility is to convert the models into a human-readable form and have them approved by cognizant system engineers (domain experts).

Conclusion

DS1 will be the first deep-space spacecraft under autonomous control. The complexity of this domain raised a number of important knowledge acquisition and representation issues, some of which we were able to address and some of which remain open. Issues were also raised by the fast-paced spiral development cycle, the embedding of the planner within the autonomy architecture, and the risk-management requirements of the space flight domain.

These issues are not unique to DS1, and are likely to occur on other projects that require autonomous control of a complex environment. As the role of autonomy increases in space exploration and other areas, so will the importance of finding good solutions to these issues.

Acknowledgments

This paper describes work partially performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract from the National Aeronautics and Space Administration. This work would not have been possible without the extraordinary effort and dedication of the rest of the Remote Agent Planning Team: Steve Chien, Charles Fry, Sunil Mohan, Paul Morris, Gregg Rabideau, and David Yan.

References

Barry Boehm. A Spiral Model of Software Development and Enhancement. *Computer*, pages 61–72, May 1988.

Michael
Compton, Helen Stewart, Vinod Baya, Martha Del Alto, Bob Kanefsky, and Jason Vincent. Electronic collaboration for the New Millenium: Internet-based Tools and Techniques for Sharing Information. In *http://ic-www.arc.nasa.gov/ic/projects/nmp-doc/nmp-doc-pres.pdf*, 1997.

Per Cederqvist et.al. Concurrent Versions System. In *http://www.loria.fr/ molli/cvs-index.html*, 1996.

Oren Etzioni. Acquiring Search Control Knowledge via Static Analysis. *Artificial Intelligence*, 62, 1993.

G.M.Brown, D.Bernard, and R.Rasmussen. Attitude and Articulation Control for the Cassini Spacecraft. a fault tolerance overview. In *14th AIAA/IEEE Digital Avionics Conference*, 1995.

Barbara Hayes-Roth. An Architecture for Adaptive Intelligent Systems. *Artificial Intelligence*, 72, 1995.

IEEE. *Proceedings of the IEEE Aerospace Conference*, Snowmass, CO, 1997.

Sanford Krasner and Douglas E. Bernard. Integrating Autonomy Technologies into an Embedded Spacecraft System—Flight Software System Engineering for New Millennium. In *Proceedings of the IEEE Aerospace Conference* (1997).

Steven Minton. Automatically configuring constraint satisfaction programs: A case study. *Constraints*, 1(1), 1996.

Nicola Muscettola, Ben Smith, Charles Fry, Steve Chien, Kanna Rajan, Gregg Rabideau, and David Yan. On-Board Planning for New Millennium Deep Space One Autonomy. In *Proceedings of the IEEE Aerospace Conference* (1997).

Nicola Muscettola. HSTS: Integrating planning and scheduling. In Mark Fox and Monte Zweben, editors, *Intelligent Scheduling*. Morgan Kaufmann, 1994.

Barney Pell, Douglas E. Bernard, Steve A. Chien, Erann Gat, Nicola Muscettola, P. Pandurang Nayak, Michael D. Wagner, and Brian C. Williams. A Remote Agent Prototype for Spacecraft Autonomy. In *Proceedings of the SPIE Conference on Optical Science, Engineering, and Instrumentation*, 1996.

Barney Pell, Erann Gat, Ron Keesing, Nicola Muscettola, and Ben Smith. Plan Execution for Autonomous Spacecraft. In Louise Pryor, editor, *Procs. of the AAAI Fall Symposium on Plan Execution*. AAAI Press, 1996.

Barney Pell, Douglas E. Bernard, Steve A. Chien, Erann Gat, Nicola Muscettola, P. Pandurang Nayak, Michael D. Wagner, and Brian C. Williams. An Autonomous Spacecraft Agent Prototype. In *Proceedings of the First International Conference on Autonomous Agents*. ACM Press, 1997.

M. Tambe, W. Lewis Johnson, R. M. Jones, F. Koss, J. E. Laird, Paul S. Rosenbloom, and K. Schwamb. Intelligent Agents for Interactive Simulation Environments. *AI Magazine*, 16(1):15–39, Spring 1995.

Brian C. Williams and P. Pandurang Nayak. A model-based approach to reactive self-configuring systems. In *Procs. of AAAI-96*, pages 971–978, Cambridge, Mass., 1996. AAAI Press.

Brian C. Williams and P. Pandurang Nayak. Immobile Robots, AI in the New Millennium. *AI Magazine*, Fall, 1996.

Toward the Development of Robust Scheduling Tools

Stephen F. Smith
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213 sfs@cs.cmu.edu

Abstract

In this note, I summarize ongoing research in the Intelligent Coordination and Logistics Laboratory (ICLL) at Carnegie Mellon University toward the development of robust scheduling tools. Three inter-related but complementary threads of research are considered: (1) the development of delayed-commitment scheduling algorithms which enable generation of schedules that are less sensitive to executional uncertainty, (2) the development of architectures that support incremental, mixed-initiative scheduling and rescheduling, and (3) the development of (re)configurable scheduling tools which allow rapid construction of high-performance application systems.[1]

Introduction

In analyzing the requirements of practical scheduling systems, the issue of robustness comes up at several levels. First, one can consider the robustness of the solutions that that the system generates. Almost invariably there is some amount of uncertainty in the executing environment (and often quite a lot). To hedge against this uncertainty, it is desirable to produce solutions that retain executional flexibility where possible and leave decision-making options open. A second requirement for robustness concerns the system's ability to respond to changed constraints and assumptions. Despite any attempt to pro-actively account for uncertainty, unexpected events will occur and circumstances generally demand an ability to incrementally revise the current solution. This may be due to the associated cost of implementing solution changes, or to support controlled convergence to acceptable solutions in collaborative problem solving contexts. In either case, the ability to efficiently and flexibly resolve conflicts and exploit improvement opportunities is a crucial requirement. Finally, one can consider the robustness of a scheduling system design across different applications. Though there is commonality in requirements across domains, different scheduling problems invariably present different challenges. There may be different dominating constraints and objectives, different types of uncertainty, peculiar problem structure and so on. High performance in a particular domain depends in large part on an ability to capitalize on domain idiosyncrasies, and the ease/cost of system development thus becomes another important concern.

In the following sections, I describe three inter-related but complementary threads of ongoing research within ICLL that aimed at addressing these issues of robustness in practical scheduling system design. Work on so-called constraint-posting scheduling algorithms, which defer commitment with respect to the exact timing of activities, is discussed first. This is followed by an overview of current work on incremental, mixed-initiative scheduling (and planning) frameworks. Finally, the development of a scheduling ontology for use as a device for configuring domain models (and ultimately full application systems) is mentioned.

Constraint-Posting Scheduling Algorithms

Most typically, a scheduling problem is formulated as one of finding a consistent assignment of start times for each input (or goal) activity. However one can alternatively formulate the problem strictly as a sequencing problem. In this case, the objective is to determine and post precedence constraints between pairs of activities contending for the same resources so as to ensure that all time and capacity constraints are satisfied. Solutions generated in this way generally represent a set of feasible schedules (i.e., the sets of activity start times that remain consistent with posted sequencing constraints), as opposed to a single assignment of start times. As such, they provide a measure of robustness against executional variances.

Our work has developed a family of constraint-posting scheduling procedures, each derived from a basic constraint satisfaction search model called PCP (Precedence Constraint Posting) (Smith and Cheng 1993). Within this model, the "variables" to be assigned are ordering decisions $O(i,j,r)$ (for activities i and j contending for resource r), each with two possible "values": *i before j* or *j before i*.. The PCP search procedure utilizes estimates of sequencing flexibility as a heuristic basis for directing the search. Estimates of sequencing flexibility are used (1) to detect unconditional sequencing decisions and perform early

[1] The research reported in this paper has been sponsored in part by the National Aeronautics and Space Administration under contract NCC 2-976, by the Department of Defense Advanced Research Projects Agency under contracts F30602-95-10018 and F30602-97-20227 and the CMU Robotics Institute.

pruning of the search space and (2) as a basis for variable and value ordering at each step of the search. The effectiveness of PCP as a constraint satisfaction scheduling procedure has been demonstrated under very general representational assumptions (Cheng and Smith 1994).

Use of the basic PCP model as a basis for schedule optimization has also been explored, leading to development of the Multi-PCP procedure for minimizing schedule makespan (Cheng and Smith 1997). Experimental results with Multi-PCP have been quite impressive. Running on classical benchmark job shop scheduling problems from the Operations Research (OR) community, Multi-PCP produced solutions comparable to state-of-the-art OR approximation algorithms. Multi-PCP has also generated the best known solutions to a more idiosyncratic "hoist" scheduling problem, which requires treatment of temporal separation constraints and sequence-dependent resource setups. The reader is referred to (Cheng and Smith 1997) for further details.

A similar constraint-posting approach has also been successfully used as the basis for an experiment scheduler for a robotic chemistry work station (Aarts and Smith 1994). In this case, the ability to accommodate so-called "flexible experiment protocols" (i.e., specification of execution intervals and flexible experiment step durations as opposed to rigid temporal constraints) was shown to yield almost a 100% improvement in overall work station throughput.

One limitation of the basic PCP model, is its fall-back reliance on a back-tracking search model in circumstances where its heuristics fail to produce a feasible solution. In practice, this approach has proved to be computationally prohibitive and most work has instead relied on backtrack-free versions of PCP, which generate relaxed solutions if necessary instead of expanding the search. To provide an alternative to the basic backtrack search model (and reduce strict reliance on the power of PCP's search heuristics), recent work has investigated the development of randomized variants of basic PCP search heuristics and their use within an iterative sampling search model (Oddi and Smith 1997). The key idea here is to vary the level of randomness according to how well informed the heuristic is in a given choice context. Experimental results on complex constraint satisfaction scheduling problems have demonstrated the efficacy of this approach.

Current work focuses on extending PCP-based search procedures and heuristics to operate with more complex resource capacity models, as well as integration with complementary heuristics for resource assignment.

Incremental, Mixed-Initiative Scheduling

In practice, planning and scheduling is rarely (if ever) a one-shot generative task aimed at satisfying pre-specified constraints and objectives. It is an iterative (and ongoing) process, concerned concurrently with (1) building understanding of the evolving problem and execution state, (2) determining what the constraints and objectives are (or should be), and (3) generating and maintaining an acceptable solution. Despite this fact, most current planning and scheduling systems are based rigidly on a "specify and solve" model of user-system interaction, where the user specifies problem inputs and constraints up-front, and the back-end problem solver is then invoked to generate a solution. This model of interaction does not match the requirements of the larger planning and scheduling process. There is no persistence of decisions over time, no ability to control how solutions change in response to changed inputs or constraints, and no ability to converge to and maintain an acceptable solution in an incremental, controlled manner.

A better framework for user-system interaction follows from a view of planning and scheduling as an incremental change process. This is the approach taken in the design of Ozone, a configurable framework for mixed-initiative scheduling and planning under continuing development at CMU (Smith et. al. 1996). Within Ozone, incremental constraint-based problem solving techniques are combined with graphical solution visualization and manipulation techniques to provide a flexible, "spreadsheet-like" planning and scheduling model. The user analyzes solution elements and formulates change directives from aggregate task-oriented perspectives; the system manages the details of implementing change directives in accordance with user goals and expectations.

The need for change arises both in advance planning contexts, when solutions are found to be less than satisfactory, and to reactively respond to problems and opportunities that arise as execution proceeds. In either case, decision-making responsibility can be variably apportioned from user to system. Various graphical metaphors are used to convey essential attributes of a desired user change, including focus (which decisions must change), scope (which decisions should not), relaxable constraints (if necessary) and search biases (for use within constraint satisfying subspaces). An agenda-based controller, descended from the earlier-developed OPIS scheduling system (Smith 1994), is responsible for mapping change directives to appropriate system change procedures. The set of change procedures available are designed to provide complementary (re)optimization, conflict resolution and change localization capabilities.

The Ozone framework has been applied to develop application systems in a range of complex planning and scheduling domains. Most notable is the DITOPS transportation scheduler (Smith and Lassila 1994b, Smith et. al. 1996), which was demonstrated originally in the domain of large-scale strategic deployment and is now being adapted for transition into operation at the US Air Mobility Command as a day-to-day airlift and tanker scheduling tool. Current research focuses on elaboration of the mixed-initiative scheduling framework to support collaboration with other planning agents and on integration with advanced data visualization and exploration tools.

Configurability and Reuse in Scheduling System Design

A second major thrust of research within the Ozone project has been the development of a configurable scheduling system: a generic framework that promotes rapid construction of domain-specific tools for specific scheduling applications through reuse and extension of previously developed components. Like other recent work toward the design of more flexible scheduling systems (Smith and Lassila 1994a, LePape 1994, Pinedo & Weh 1997), Ozone has been developed with strong emphasis on object-oriented design principles, and is organized as an extensible class library. However, while reliance on object programming techniques and practices provides a foundation for achieving configurability in scheduling system design, a class library by itself does not necessarily promote configurability. Without a higher-level, conceptual model for configuring scheduling applications, reuse of system components and class libraries tends to be quite difficult for anyone other than the original system designers.

Within Ozone, this conceptual model for scheduling system design is achieved by first committing to basic architectural constraints, which provides a structure for identifying and decomposing functional components, and then super-imposing a mechanism for mapping application characteristics and requirements to implied functional components. The first step can be seen as a domain specific counterpart to the use of architectural patterns in software engineering (Gamma et. al. 1994). As already mentioned in the previous section, the Ozone system architecture derives from commitment to a constraint-based scheduling model.

The second step integrates these ideas with recent trends in artificial intelligence toward (1) treating knowledge engineering and knowledge acquisition as a modeling activity (Wielinga et. al. 1992), and (2) development and use of ontologies as a basis for knowledge sharing and reuse (Gruber 1993). The Ozone scheduling ontology (Smith and Becker 1997) defines a meta-model of the scheduling domain. It provides a vocabulary for describing those aspects of the scheduling domain that are relevant to construction of an application system, and a set of constraints on how concepts in the ontology fit together to form consistent domain models. Consistency, in this context, relates to the information and knowledge required to insure executability of the model. Generally speaking, the Ozone ontology serves to map user-interpretable descriptions of an application domain to application system functionality.

This linkage is established through the inclusion of *capabilities* as an integral part of the definition of concepts in the ontology. Capabilities designate encapsulated behaviors that are intrinsic to various domain concepts. These behaviors refer to specific software components in the underlying class library. As a result, the concepts in the ontology can be seen as the actual building blocks for composing and customizing constraint-based solution methods. For example, the concept of a "resource" contributes capabilities for querying and managing its available capacity over time, and different resource types (e.g., reusable, consumable) provide specific "implementations". Given a solution method that incorporates these capabilities, the ontology provides a direct basis for its customization to match the resources in any target domain. It promotes rapid configuration of executable systems and allows concentration of modeling effort on those idiosyncratic aspects of the target domain.

Illustrating the potential of this approach, Ozone was recently applied to develop a system for reactive re-planning of areo-medical evacuation missions in a period of just two person months (Lassila et. al. 1996). Current research focuses on extension of the Ozone ontology to encompass scheduling methods, and on development of interactive, model building tools.

Acknowledgements

The work reported in this paper is the result of the collective efforts of many individuals, including Robert Aarts, Marcel Becker, Eric Biefeld, Stephen Chen, Casper Cheng, Ora Lassila, Dirk Lemmermann, Angelo Oddi, Gary Pelton, Mark Shieh and Ben Werle.

References

Aarts, R. and Smith, S.F. 1994. A High Performance Scheduler for an Automated Chemistry Workstation. In *Proceedings 1994 European Conference on Artificial Intelligence*, Amsterdam

Cheng, C., and Smith, S.F. 1997. Applying Constraint Satisfaction Techniques to Job Shop Scheduling. *Annals of Operations Research* 70:327-357.

Cheng, C. and Smith, S.F. 1994. Generating Feasible Schedules under Complex Metric Constraints. In *Proceedings 12th National Conference on Artificial Intelligence*, Seattle, WA.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. 1994. *Design Patterns: Elements of Reusable Object-Oriented Design*. Addison-Wesley.

Gruber, T.R. 1993. Towards Principles for the design of Ontologies used for Knowledge Sharing, Technical Report KSL-93-04, Stanford University, Palo Alto, CA, Aug.

Lassila, O., Becker, M. and Smith, S.F. 1996. An Exploratory Prototype for Aero-Medical Evacuation Planning, CMU Robotics Institute Technical Report CMU-RI-TR-96-02, January.

Le Pape, C. 1994. Implementation of Resource Constraints in ILOG schedule: A Library for the Development of Constraint-based Scheduling *Intelligent Systems Engineering*.

Oddi, A. and Smith, S.F. 1997. Stochastic Procedures for Generating Feasible Schedules. In *Proceedings 14th National Conference on Artificial Intelligence*, Providence RI, AAAI Press.

Pinedo, M. and Weh, B. 1997. On the Design of Object-Oriented Scheduling Systems. *Annals of Operations Research*, 70.

Smith, S.F. 1994. OPIS: A Methodology and Architecture for Reactive Scheduling. In *Intelligent Scheduling* (eds. Zweben and Fox), Morgan Kaufmann.

Smith, S.F. and Cheng, C. 1993. Slack-Based Heuristics for Constraint Satisfaction Scheduling. In *Proceedings 11th National Conference on Artificial Intelligence*, Washington DC.

Smith, S.F., and Lassila, O. 1994a. Configurable Systems for Reactive Production Management. *In Knowledge-Based Reactive Scheduling*, IFIP Transactions B-15, North-Holland, Amsterdam (The Netherlands).

Smith, S.F. and Lassila, O. 1994b. Towards the Development of Flexible Mixed-Initiative Scheduling Tools. In *Proceedings ARPA/Rome Laboratory Planning Initiative Workshop*. Tucson AZ.

Smith, S.F., Lassila, O. and Becker, M. 1996. Configurable Systems for Mixed-Initiative Planning and Scheduling. In *Advanced Planning Technology*. (ed. A. Tate), AAAI Press.

Smith, S.F. and Becker, M. 1997. An Ontology for Constructing Scheduling Systems. In *Proceedings 1997 AAAI Spring Symposium on Ontological Engineering*. AAAI Press Technical Report, forthcoming.

Wielinga, B., Velde, W.V., Schreiber, G., and Akkernamans, H. 1992. The KADS Knowledge Modeling Approach. In *Proceedings 2nd Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop*. Hitachi Advanced Research Laboratory

Solar System Ephemeris and Target Visibility Processing for HST

Scott Stallcup

Space Telescope Science Institute
3700 San Martin Dr.
Baltimore MD, 21218

Abstract

The "Percy" computer program provides ephemeris and geometrical event information about solar system objects. Percy is the primary component of the Moving Object Support System (MOSS) at the Space Telescope Science Intstitute. Percy was designed specifically for Hubble Space Telescope observation planning, though it should be a useful planning tool for any ground or space based observatory. An overview of the features and capabilities of Percy is presented.

Introduction

The Hubble Space Telescope Moving Object Support System consists of just one primary computer program called "Percy". Percy is an interactive program which provides ephemeris and geometrical event information about solar system objects. These objects include the Sun, major planets and their natural satellites, comets, and asteroids.

Percy performs four primary functions :

- Locates time spans during which complex geometric constraints are satisfied.
- Graphically displays a field of view from any vantage point.
- Generates text reports of solar system phenomena
- Generates ephemeris files.

This paper presents a very brief introduction to the Percy computer program. The major features are discussed through example Percy commands.

Percy, named after Percival Lowell, and a companion program, Clyde, named after Clyde Tombaugh, were originally developed by the Jet Propulsion Laboratory (JPL) under contract to NASA Goddard Spaceflight Center and the Space Telescope Science Institute (STScI). STScI took over the development and maintenance of Percy and Clyde in 1992. Since then, extensive modifications and improvements have been made including a port from VAX/VMS to SPARC Solaris. The functionality of Clyde was merged into Percy, so Clyde is no longer supported.

Percy was originally developed specifically for Hubble Space Telescope (HST) observation planning. While it contains some HST specific features, Percy should be useful for almost any ground or spacecraft based observing system.

Internally, Percy uses the SPICELIB software library supplied by JPL's Navigation and Ancillary Information Facility (NAIF). SPICELIB provides many lower level services, calculations, and ephemeris file utilities.

A Percy configuration is a small set of data files which defines ephemerides and other solar system data. The details of these files are hidden from most Percy users. Advanced users may need to modify the configuration for some observation planning sessions. Configuration data files include :

- NAIF supplied solar system body ephemeris files (.bsp files).
- Spacecraft ephemeris data files.
- Files that contain body size, shape, prime meridian, and rotational information.
- Leap Second data.
- Star Catalogs for graphical displays. This includes the SAO and HST Guide Star Catalogs.
- Telescope aperture data for graphical display.

Percy is a command line system with a custom built-in scripting language. Commands can be entered interactively or through the execution of nested procedure files. The language is free formatted and semi-colon terminated. It also supports symbol definition and substitution.

The Percy language allows for the flexible entry of dates and times in many common time formats. By default all input times are assumed to be in UTC. Numerical values can be entered using many common units. However, the

default units are kilometers, seconds (time), and arc-seconds (angular measure).

Program output can be directed to the computer screen or to data files. Graphics output can be directed to color PostScript files or to X/Motif devices.

Before discussing the major features of Percy, a few key concepts must presented. The next section presents basic information about time and window manipulation concepts. A later section describes solar system body conventions.

Time, Windows, and Intervals

Every Percy session has a set of default time bounds. These bounds represent the default time limits of all commands. The SET BOUNDS command is used to set the default time bounds until the end of the Percy session. For example, to set the session bounds to the entire year of 1997, enter :

```
SET BOUNDS FROM 1 JAN 1997
TO 1 JAN 1998;
```

A key Percy idea is the concept of a "window". A window is a finite collection of disjoint (non-overlapping) time intervals. For example, the following set of three time intervals forms a window :

```
From 1 JAN 1997 12:00:00
To   2 JAN 1997 03:00:00

From 4 JAN 1997 01:00:00
To   5 JAN 1997 11:00:00

From 9 JAN 1997 01:00:00
To   9 JAN 1997 01:02:00
```

Percy contains an extensive set of commands to create and manipulate named windows. To create a window named TEST1 with just one interval matching the default session bounds, enter :

```
CREATE TEST1;
```

To create windows with specific time intervals, additional arguments are needed. To create a window with one interval from 12/1/97 to the end of the default bounds enter:

```
CREATE TEST2 FROM DEC 1 1997;
```

To create a window with a single interval from the default start boundary to 11/1/97 enter :

```
CREATE TEST3 TO 1 NOV 1997;
```

Finally, to create a window with a single interval from 10/15/1997 until noon on 12/15/1997, enter :

```
CREATE TEST4 FROM 10/15/1997 00:00:00
TO 12/15/1997 12:00:00;
```

Note that in the above examples several different date formats were used. Percy understands many unambiguous date representations. By default, all date-time expressions are entered and displayed in calendar format, UTC.

Percy supports several primitive window operations. Among these are the basic set operations INTERSECT, COMPLEMENT, and UNION. The union of two windows is analogous to the union of two sets. The resulting window contains every point in either of the original windows. Two windows may be unioned with the UNION command. For example, to create a new window named TEST5 which is the union of TEST1 and TEST2, enter:

```
UNION TEST1 TEST2 TEST5;
```

Two windows may be intersected with the INTERSECT command. For example, to create a new window named TEST6 which is the intersection of TEST1 and TEST2, enter:

```
INTERSECT TEST1 TEST2 TEST6;
```

The COMPLEMENT command is useful for finding intervals when some condition is not fulfilled. For example, complementing TEST2 produces the intervals within the default bounds of the planning session that are not in TEST2. The following command creates a new window TESTC2:

```
COMPLEMENT TEST2 TESTC2;
```

Many other window manipulation commands are supported. The following table briefly lists each command:

COPY	Copies one window to a new window.
CREATE	Creates a new window.
DROP	Deletes a window.
RESTORE	Loads a window from a file.
STORE	Stores a window in a file.
COMPLEMENT	Complements a window.
CONTRACT	Reduces intervals by some time x.
EXPAND	Increases intervals by some time x.
FILL	Merges adjacent intervals.
FILTER	Deletes intervals less than time x.
SHIFT	Moves intervals +/- by some time x.
DIFFERENCE	Forms intervals by diff. two windows.
INTERSECT	Intersect two windows into a third.
UNION	Union two windows into a third.
SHOW WINDOW	Display intervals in a window.
SHOW BOUNDS	Display the current session bounds.

Bodies

Percy supports three types of bodies: ephemeris bodies, designated objects, and stars. An ephemeris body is any body for which ephemerides are available. Ephemeris files for the planets and their major satellites are supplied by the NAIF group at JPL.

Ephemerides for minor satellites, asteroids, and comets may be numerically integrated from user supplied orbital elements. HST and other artificial satellites for which ephemeris data exists are also considered to be ephemeris bodies.

Ephemeris bodies for which physical size/shape data is known are drawn graphically as tri-axial ellipsoids. The physical data is also needed for some geometrical event calculations (e.g. occultation calculations).

A designated object is a temporary object, created by one of the `DESIGNATE` commands. Each designated object is specified by a set of coordinates in a particular Target Reference Frame (TRF) centered at a particular ephemeris body. Each TRF is associated with a separate `DESIGNATE` command.

Stars are only displayed by the graphics subsystem. No geometrical event calculations can reference a star. For example, Percy does not currently support operations involving star occultation events.

Percy uses integer codes instead of names to refer to ephemeris bodies for two reasons. The names of some satellites conflict with the names of some asteroids and comets. Also, some satellites are commonly referred to by names other than those approved by the IAU.

The type of a body (barycenter, planet, satellite, comet, asteroid, or spacecraft) and the system to which it belongs (Earth, Mars, Jupiter, Saturn, Uranus, Neptune, or Pluto) can be recovered algorithmically from the integer code assigned to a body.

In practice, names are defined as symbols using the Percy `DEFINE` command. For example, to define symbols for the body codes for Phobos and Deimos (natural satellites of Mars) :

```
DEFINE PHOBOS   401;
DEFINE DEIMOS   402;
```

Following the above commands, references to these bodies can be made using either the integers or the symbolic names.

Negative codes are reserved for man-made objects such as the Hubble Space Telescope, the Galileo spacecraft, etc. The body code for HST is -1.

The smallest positive codes are reserved for the Sun and the planet barycenters:

0 Solar system
1 Mercury
2 Venus
3 Earth
4 Mars
5 Jupiter
6 Saturn
7 Uranus
8 Neptune
9 Pluto
10 Sun

The code for a natural satellite is normally computed by adding its IAU designation to 100 times the code for its barycenter. For example, Ananke, the 12th satellite of Jupiter (JXII), is body number 512. Some examples are listed below :

301 Moon
401 Phobos
402 Deimos
501 Io
502 Europa
503 Ganymede
504 Callisto
801 Triton
802 Nereid
901 Charon

A planet is always considered to be the 99th satellite of its own barycenter. For example, Jupiter is body number 599.

199 Mercury
299 Venus
399 Earth
499 Mars
599 Jupiter
699 Saturn
799 Uranus
899 Neptune
999 Pluto

Numbers between 1001 and 1099 are assigned to designated objects as the objects are created. These numbers may not be used for ephemeris bodies.

Comets are numbered in the range [1000000, 2000000).
Asteroids are numbered in the range [2000000, 3000000).

FIND Commands

Percy supports several FIND commands that find windows where some geometric constraint is satisfied. For example, a single FIND command can find the intervals when the Jovian satellite Io is occulted by Jupiter as viewed from Earth. The actual Percy command to do this over the session bounds could be :

```
FIND OCC OCCULTATION OF IO BY JUPITER
FROM EARTH;
```

Where OCC is the name of a window containing the intervals requested. IO, JUPITER, and EARTH are Percy symbols. The integer body codes could be used instead.

Eclipses and transits can also be found similarly :

```
FIND ECL ECLIPSE OF IO BY JUPITER FROM
EARTH;

FIND TRN TRANSIT OF JUPITER BY IO FROM
EARTH;
```

Window manipulation commands can be used to create a window named NOT_OBSCURED where Io is not "obscured" by Jupiter :

```
UNION      OCC       ECL OBSCURED;
UNION      OBSCURED  TRN OBSCURED;
COMPLEMENT OBSCURED -    NOT_OBSCURED;
```

Another FIND command could be used to find a window named PLAN_WIND within window NOT_OBSCURED when Io is at least 50 degrees from the Sun as viewed from the Earth :

```
FIND PLAN_WIND SEPARATION OF IO SUN
FROM EARTH GREATER THAN 50 DEGREES
WITHIN NOT_OBSCURED;
```

To summarize, the sequence of commands listed above in this section can be used to find time intervals under specific constraints. In this case, the constraints are that Io is not occulted by Jupiter, transiting Jupiter, eclipsed by Jupiter, and is greater than 50 degrees from the Sun. The time intervals for these conditions are in window PLAN_WIND.

Note that the WITHIN clause can often provide a significant performance improvement because it limits the time span of the FIND calculation to intervals previously calculated.

Note also the keyword DEGREES. The default for all angular quantities is ARCSECONDS.

Finally note that, other constraints in addition to GREATER THAN 50 DEGREES could have been specified such as LESS THAN 75 DEGREES.

Windows under the following constraints may be found using the FIND command:

ANGULAR RATE	The angular rate of a target body across the sky satisfies a numeric constraint.
APPARENT DIAMETER	The apparent angular diameter of a target body satisfies a numeric constraint.
CENTRAL MERIDIAN	The longitude of the sub-observer point on a target body is confined to a particular arc.
DISTANCE	The apparent distance between a pair of bodies satisfies a numeric constraint.
ECLIPSE	One target body is eclipsed by another.
ELONGATION	The angular separation of a target body from the Sun satisfies a numeric constraint.
MOONLIGHT	The Moon is visible from an Earth orbiting spacecraft.
OCCULTATION	One target body is at least partially occulted by another.
ORBITAL LONGITUDE	The orbital longitude of a target body is confined to a particular arc.
PHASE	The phase angle of a target body satisfies a numeric constraint.
QUADRATURE	The elongation of a target body is 90 degrees.
RANGE RATE	The range rate between a pair of target bodies satisfies a numeric constraint.
SAA	An Earth orbiting spacecraft is inside the South Atlantic Anomaly.
SEPARATION	The angular separation of a target body from the Sun satisfies a numeric constraint.

TRANSIT	One target body is in transit across another (full disc).
UMBRA	An Earth orbiting spacecraft is inside the (umbral) shadow of Earth.

Reports

A report is a tabular collection of function values over time. The values within a row all correspond to the same epoch. The values within a column all correspond to the same function. The functions are not necessarily numeric functions. (For example, a "Boolean" function that states whether or not an object is visible as a function of time.)

The following example illustrates how to create a report that gives the right ascension and declination of the Moon as seen from Earth as a function of time.

```
NEW ENTRY MOON_RA DESCRIPTION J2000
RIGHT ASCENSION OF 301 FROM 399
UNITS DEGREES      FORMAT X.XXXXXXX
HEADING "R.A.";

NEW ENTRY MOON_DEC DESCRIPTION J2000
DECLINATION OF 301 FROM 399
UNITS DEGREES      FORMAT X.XXXXXXX
HEADING "Dec.";

NEW ENTRY TIME DESCRIPTION TIME FORMAT
UTC HEADING "Time" NO LEGEND;

NEW REPORT MOON_POSITION ENTRIES
           TIME MOON_RA MOON_DEC;

GENERATE SMART REPORT MOON_POSITION
         FROM   15 MAR 1991
         TO     25 MAR 1991
         EVERY  1 DAY
         HEADER LEGEND;
```

The NEW ENTRY commands define the columns in the report. The NEW REPORT command defines the columns to put into the next report. The GENERATE SMART REPORT command specifies the time range, header, and report columns for the printed report. The output from these commands is :

```
R.A. --- J2000 RIGHT ASCENSION OF 301
         FROM 399 UNITS DEGREES
         FORMAT X.XXXXXXX
Dec. --- J2000 DECLINATION OF 301
         FROM 399 UNITS DEGREES
         FORMAT X.XXXXXXX

Corrections for stellar aberration are
enabled.
```

```
Time          R.A.          Dec.
(utc)         (degrees)     (degrees)
=======================================
1991 MAR 15    3.380E+02    -5.6302818
1991 MAR 16    3.499E+02     0.1388443
1991 MAR 17    2.0758449<*   6.0211735
1991 MAR 18   14.7290874    11.7237596
1991 MAR 19   28.0758534    16.9129390
1991 MAR 20   42.2451251    21.2271872
1991 MAR 21   57.2206409    24.3105445
1991 MAR 22   72.7833123    25.8708325
1991 MAR 23   88.5284585    25.7492790<
1991 MAR 24    1.040E+02 *  23.9659817
1991 MAR 25    1.188E+02    20.7099431
```

The "*" is used to mark changes of magnitude. The ">" is used to mark points where a quantity begins increasing. The "<" is used to mark points where a quantity begins decreasing.

Satellites, Comets, and Asteroids

Percy can generate NAIF formatted ephemeris files (.BSP files) for satellites, comets, and asteroids using two body numerical integration. These ephemeris files can then be used in FIND commands, reports, or other Percy commands.

Ephemeris files are created using the MAKE SATELLITE, MAKE ASTEROID, or MAKE COMET commands. These commands accept an epoch and a set of orbital elements in various formats. The ephemeris file is generated relative to a user specified time span.

The following example creates an asteroid for the month of March 1990 using body number 10001. The file is SAMPLE.BSP.

```
MAKE ASTEROID 10001 USING ELEMENTS
SEMI-MAJOR AXIS          294123638.3 KM
ECCENTRICITY             0.51376
INCLINATION              21.289 DEGREES
LONGITUDE OF NODE        173.213 DEGREES
ARGUMENT OF PERIAPSE     11.59 DEGREES
MEAN ANOMALY AT EPOCH    49.10 DEGREES
EPOCH OF ELEMENTS        21 AUG 1989
BEGINNING                1 MAR 1990
ENDING                   1 APR 1990
FRAME                    EQUATORIAL
EQUINOX                  J2000
MAXIMUM ABSOLUTE ERROR   500 KM
INSERT INTO NEW          SAMPLE.BSP;
```

The following example creates a comet for the months of June, July, and August 1989 using body number 10001. The file is RACKNE.BSP.

```
MAKE COMET 10001 USING ELEMENTS
DISTANCE OF PERIAPSE     294123638.3 KM
ECCENTRICITY             0.51376
INCLINATION              21.289 DEGREES
LONGITUDE OF NODE        173.213 DEGREES
ARGUMENT OF PERIAPSE     11.598 DEGREES
TIME OF PASSAGE          1 JUN 1989
EPOCH OF ELEMENTS        1 SEP 1989
BEGINNING                1 JUN 1989
ENDING                   1 SEP 1989
FRAME                    EQUATORIAL
EQUINOX                  J2000
MAXIMUM ABSOLUTE ERROR   500 KM
RADIAL NONGRAV           7.01E-10
TANGENTIAL NONGRAV       1.549E-10
INSERT INTO NEW          RACKNE.BSP;
```

The following example creates a satellite of Venus for the months of June, July, and August 1989 using body number 201. The file is SAT.BSP.

```
MAKE SATELLITE 201 ORBITTING VENUS
USING ELEMENTS
SEMI MAJOR AXIS          29438.3 KM
MEAN LONGITUDE AT EPOCH  219.31 DEGREES
MEAN MOTION              419 DEGREES/DAY
EPOCH OF ELEMENTS        21 AUG 1994
INCLINATION              31 DEGREES
BEGINNING                1 JUN 1989
ENDING                   1 SEP 1989
INSERT INTO NEW          SAT.BSP;
```

User Ephemeris File Generation

NAIF formatted ephemeris files are often not in an ideal form for use by other software systems. Percy supports the generation of two types of ephemeris files. The first file is human readable and in the traditional form where state vectors are written one per line. Each line contains the time followed by the Cartesian position and velocity vectors. The ephemeris for a body can be generated relative to any other body with or without light-time correction. The Percy TRACK command generates these files.

The second form of ephemeris file contains a piecewise Chebychev polynomial representation. This representation is limited to ephemeris data that is relative to the Earth and light-time corrected. These files are not human readable, but they are portable to any computer platform. The Percy CHEBY command generates these files. Applications using CHEBY generated files must call Percy C++ software to determine state vectors for the body.

The following TRACK command generates a file relative to Earth for Io. The ephemeris is generated over the window named A_WIND. The output filename is A_FILE.TRH. The data (not light-time corrected) is generated every 5 minutes.

```
TRACK IO RELATIVE EARTH OVER A_WIND
EVERY 5 MINUTES INTO A_FILE;
```

The following command adds light time correction. The command also generates points such that linear interpolation between the points yields an accuracy of 0.1 arcseconds.

```
TRACK IO RELATIVE EARTH OVER A_WIND TO
0.1 ARCSECONDS INTO A_FILE CORRECT FOR
LIGHT TIME;
```

The following CHEBY command generates a Chebychev polynomial ephemeris file. This example generates a file named A_FILE.CBY which essentially contains the same information as that generated in the last TRACK command.

```
CHEBY IO FROM 1 JAN 1990 TO 1 FEB 1990
INTO A_FILE ANGULAR_ERROR 0.1;
```

Graphics

Percy can generate instantaneous wireframe graphical displays of the solar system. Currently, Percy supports graphical display to X/Motif devices and color PostScript files. At a minimum, given a time, a pointing, a vantage point, and a field of view, Percy can draw the scene that would be visible from a telescope. For example, a scene of Jupiter, its satellites, and background stars can be plotted as viewed from HST. HST apertures can also be included in the field of view.

The image plane of the telescope is centered at the optic axis. Points in the image plane are specified by their coordinates along axes labeled V2 and V3. Percy normally draws scenes as you would see them in images returned from the telescope. V2 is normally positive to the right, V3 is positive up, and V1 is positive toward you. V2V3 values are normally measured in arc seconds. The V1 axis lies along the boresight of the telescope.

The contents of a scene depend primarily on the current time, location (vantage), and orientation (pointing) of the telescope. The term "roll angle" denotes the spherical angle, measured at the telescope +V1 axis, between the reference axis vector and the +V3 telescope axis. For example; with celestial north as the reference vector, roll angle is the north position angle, at +V1, of the +V3 axis. By convention, roll angle is measured counterclockwise, through east.

By default, the telescope assumes a "nominal" roll. This means that the +V3 axis is placed such that the sun lies in the V1/+V3 half-plane. The roll angle can be changed by the SET ROLL command.

The term "orientation" denotes the celestial orientation of the sky with respect to the graphics device. For example; a

North-up orientation is one in which celestial north lies in the "up" direction on the screen. By default, the sky is presented in a North-up orientation. The orientation can be changed by the SET ORIENTATION command.

The term "instrument" denotes the position (in V2/V3 coordinates) of the telescope field-of-view which overlays the pointing position. This will determine the apparent positions of the overlays with respect to the sky. By default, the instrument position is the +V1 axis; i.e. the center of the field-of-view (V2 = V3 = 0). However, an arbitrary field-of-view location can be placed over the object of interest via the SET INSTRUMENT command.

The term "center" denotes that location on the sky, in celestial coordinates (e.g. right ascension, declination) which corresponds to the center of the graphic device. By default, the position specified by the SET POINTING command becomes the center of the plot. However, the center position may be changed by use of the SET CENTER command.

The Percy commands to generate graphical images are listed below :

SET MOTIF	All commands that follow are to generate graphics to an X/Motif window.
SET POSTSCRIPT	All commands that follow are to generate graphics to a PostScript file.
SET TIME	Set the time for the next image.
SET POINTING	Set the direction to "look". Could be a body.
SET VANTAGE	Set the position to view from. Could be a body.
SET FIELD OF VIEW	Sets the field of view angle.
SET ROLL	Sets the roll angle of a spacecraft.
SET ORIENTATION	Sets the orientation of a spacecraft.
SET INSTRUMENT	Sets the instrument or position within the field of view to place target.
SET CENTER	Defines the center of a plot relative to the graphics device.
SET ATTRIBUTE	Sets specific colors or other characteristics for a graphics object.
SET PICTURE	If more than one image is to be plotted, this command identifies the next image to be updated.
SET MAGNITUDE LIMITS	Sets the magnitude range for the display of stars.
SET OVERLAYS	Identifies one or more graphics overlays to be displayed. Overlays can be used to indicate grid lines or telescope apertures.
DRAW	Executes the drawing operation for the currently defined scene.

The following sequence of commands illustrates the typical sequence of commands. Four pictures are created. The first centers a wide field of view on Jupiter. The others center a much narrower field of view on Io, Europa, and Ganymede.

```
SET MOTIF;
SET TIME 1 JAN 1997;
SET FIELD OF VIEW 4 ARCMINUTES;
SET POINTING AT JUPITER
SET PICTURE 1 OF 4 ;
DRAW;

SET FIELD OF VIEW 2 ARCSECONDS;
SET POINTING AT IO;
SET PICTURE 2 OF 4;
DRAW;

SET POINTING  AT EUROPA;
SET PICTURE 3 OF 4;
DRAW;

SET POINTING  AT GANYMEDE;
SET PICTURE 4 OF 4;
DRAW;
```

Remarks

Percy is a robust tool for the planning of ground or space based observations of solar system objects. Percy was originally developed specifically for HST observation planning. While it contains some HST specific features, Percy should be useful for almost any ground or spacecraft based observing system. Percy is planned to be used for the Far Ultraviolet Spectrographic Explorer (FUSE) spacecraft.

This paper has presented a very brief introduction to the Percy program. Percy supports many features, commands, and command options that were not mentioned here. See the reference below for more information.

Reference

Complete documentation for Percy can be found on the World Wide Web in:

http://www.pst.stsci.edu/moss/percy_main.shtml

Multiagent Learning for Autonomous Spacecraft Constellations

Peter Stone
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
pstone@cs.cmu.edu
http://www.cs.cmu.edu/~pstone

Abstract

Achieving spacecraft autonomy, the goal of the NASA New Millennium Project, promises to increase the range of possible space missions while drastically reducing costs. However, *individual* spacecraft autonomy with *fixed* control algorithms is insufficient for meeting New Millennium goals. This paper presents the *layered learning* technique for flexible control in complex, real-time, multiagent environments. Layered learning has been successfully implemented in one such domain. Here, potential space-related applications are presented.

Introduction

Achieving spacecraft autonomy, the goal of the NASA New Millennium Project, promises to increase the range of possible space missions while drastically reducing costs. However, *individual* spacecraft autonomy with *fixed* control algorithms is insufficient for meeting New Millennium goals.

This paper presents multiagent learning techniques and flexible teamwork structures which are being developed in noisy, real-time environments. These techniques are applicable towards the realization of adaptivity, autonomy, and multiagent plan refinement in *constellations* of spacecraft. This new approach, multiagent *layered learning*, is a hierarchical, bottom-up method that makes use of expert domain decompositions to seamlessly integrate multiple learning modules.

A preliminary investigation and development of this strategy in a complex, noisy domain that requires real-time interleaving of planning and execution has shown promising empirical results in flexibly acquiring and using the learned knowledge at different levels of individual and team behaviors (Stone and Veloso 1998).

A specific potential space-related application is the control of constellations of interferometer-bearing spacecraft on long-range mapping missions. As well as increasing the range of possible missions and the size of possible fleets, spacecraft autonomy could reduce mission operations costs, exclusive of data analysis, by up to 60% (Ridenoure 1995), with even larger savings likely for multi-craft autonomy.

Significant multiagent learning challenges to be met in this endeavor include developing communication protocols among the spacecraft; developing decision-making methods that facilitate real-time reactions to unexpected opportunities and/or problems; coordination of multiple spacecraft in pursuit of a common goal; feedback monitoring systems to evaluate performance; and learning methods to adapt behavior over certain control variables. Since constellations may include different types of spacecraft, the organization of the spacecraft into different mission roles is an important task. Particularly in the event of spacecraft failure, it would be desirable to have a constellation that is capable of reorganizing itself in an appropriate way.

The rest of the paper is organized as follows. The next section presents layered learning, the general research methodology expounded in this paper. The following section presents space-related application areas in which this layered learning can be used. Then previous and current research regarding layered learning is summarized. The penultimate section presents related work and the final section concludes.

Methodology

Along with distributed techniques studied by all researchers in Multiagent Systems, space domains require real-time control in noisy environments. As of yet, there has been little work with multiagent systems in such complex domains. Because of the inherent complexity of this type of multiagent system, Machine Learning is an interesting and promising area to merge with Multiagent Systems. Machine Learning has the potential to provide robust mechanisms that leverage upon experience to equip agents with a large spectrum of behaviors, ranging from effective individual performance in a team, to collaborative achievement of independently and jointly set high-level goals.

The research presented here focuses on learning in this particularly complex class of multiagent domains. The principal question to be answered is

Can agents learn to work together in a real-time, noisy environment?

To be precise, the prospects of using machine learning

techniques to improve an agent's behavior are examined in domains with the following characteristics:

- several independent agents with the same well-defined high-level goal;
- a need for real-time decision-making;
- sensor and actuator noise.

The agents are assumed to have at their disposal the following resources:

- sensors of the environment that give partial, noisy information;
- the ability to process the sensory information and use it to update a world model;
- noisy actuators that affect the environment;
- low bandwidth communication capabilities;
- a low-level protocol for communication;

As a basis for learning at the multiagent level, agents are also organized into a flexible team structure that allows them to dynamically adjust their overall formation as well as their individual roles within the formations.

It is assumed that, due to the complexity of the environment, agents in domains with the above characteristics are unable to learn effective direct mappings from their sensors to their actuators, even when saving past states of the world. Thus, the approach taken is to break the problem down into several behavioral layers and to use machine learning techniques when appropriate. Starting with low-level behaviors, the process of creating new behavior levels and new machine learning subtasks continues until high level strategic behaviors that take into account both teammate and opponent strategies are reached. At every behavior level, the end product should be a method for choosing an action every time the agent gets sensory information. The appropriate behavior granularity and the aspects of the behaviors to be learned will be determined a priori as a function of the specific domain. This new approach is called *layered learning* (see Figure 1).

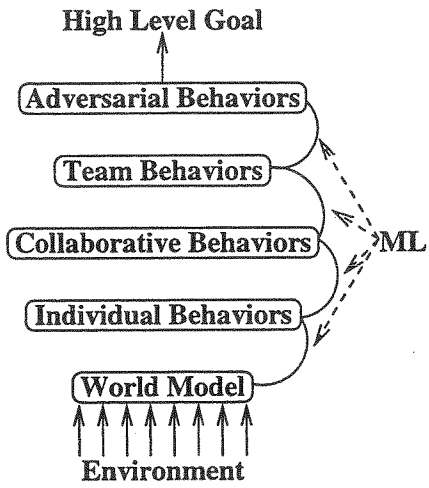


Figure 1: An overview of the layered learning framework. It is designed for use in domains that are too complex to learn a mapping straight from sensors to actuators. It uses a hierarchical, bottom-up approach. Not all multiagent domains require adversarial behaviors on top of the team behaviors.

Application Areas

For missions involving constellations of spacecraft to be truly autonomous, *distributed, adaptive, autonomous control* is necessary. Instead of centralized control of spacecraft with fixed control algorithms, methods are required for the coordination of adaptive spacecraft in a distributed fashion. Centralized control of the entire constellation from a single craft risks complete mission failure if just a single craft fails. Furthermore, distributed control facilitates variable fleet sizes, possibly up to a hundred spacecraft for a single mission (Chien 1996). The ability for autonomous spacecraft to learn, or adapt to their environments, would allow mission profiles to be framed more loosely and to apply to a wider range of environments (Chien 1996).

Interferometry for Imaging Distant Objects

A particularly good example of the potential benefits of multiagent learning techniques to space-related problems is interferometry missions for imaging distant objects[1]. As described in (Chien 1996), consider

> a network of interferometer bearing spacecraft in an earth-orbiting configuration, using solar power, with limited data storage, and an interferometer for imaging distant objects (such as planets around nearby stars).

In this scenario, distributed, adaptive techniques have several potential benefits:

- If any subset of the spacecraft fail or temporarily must devote their resources to spacecraft safety needs, the remaining spacecraft could continue performing interferometric mapping. Although they may need to shift to a different target, valuable observation time would not be wasted.
- If an unexpected scientific observation opportunity arises, some or all of the constellation could shift attention to the unexpected event. Lower priority goals could be handled with remaining resources or deferred.
- If any of the spacecraft in the constellation begins degrading such that it is still operational, but with limited capacities or increased time needs for certain activities, an adaptive planner could reconfigure the

[1]For example, the New Millennium Deep Space 3 Mission is a constellation of interferometer bearing spacecraft.

constellation so as to make optimal use of each of the spacecraft to maximize interferometric mapping capabilities.

Notice that in this application area the constellation must be able to handle *noisy* real-world data about spacecraft state and imaging conditions. Furthermore, the constellation should be able to respond to unexpected events quickly, or in *real-time*. Otherwise opportunities may pass before they are seized.

Other Opportunities

Several other potential multiagent learning applications exist in real-time, noisy situations:

- As NASA moves towards more frequent missions, the frequency of inter-space meetings between multiple shuttles and space stations will also increase. Distributed scheduling is useful (i) so that the mission is robust to the failure of any individual spacecraft (i.e. the one with the scheduler), and (ii) to avoid the necessity of forwarding large amounts of local data to a centralized source. However, if each shuttle and each station is equipped with its own scheduler, then there will need to be methods for these schedulers to coordinate their plans so that all constraints and goals can be met.
- Within a complex system such as a space station, the same distributed, adaptive systems can offer the same advantages as they do for interferometric missions. If one subsystem fails or degrades, appropriate and timely reconfigurations of goals and/or resource allocations may be necessary.
- On planetary missions, it may be desirable to deploy several rovers to explore, possibly out of radio range from a mothership. Again, multiagent learning in real-time, noisy situations could increase the effectiveness or enable the eventual success of such missions.

Previous Work

Current and past research on layered learning is primarily in the emerging domain of simulated Robotic Soccer (Kitano *et al.* 1997). An advantage of using this domain instead of immediately developing techniques in a space-related application area is that the simulator already exists and it is being used by several researchers around the world. Thus competing systems are being developed elsewhere, which makes realistic testing particularly accessible.

Layered learning has already been successfully implemented in the robotic soccer domain (Stone and Veloso 1998). First, clients (i.e. programmed players) use a neural network to learn to intercept a moving ball. Second, clients use a decision tree to learn the likelihood of completing a pass to a given teammate. A key aspect of these two behaviors is that the clients use the trained neural network when attempting to receive the ball during decision tree training (Stone and Veloso 1998). Such interaction of learned behaviors at different behavior levels is essential to layered learning.

Additional high-level learned behaviors are also being developed. Clients can use the output of the decision tree to choose whether to pass, shoot, or dribble the ball in a given situation. They can also learn to position themselves effectively on the field, both individually and as a team.

These positioning behaviors entail strategic reasoning to counteract opponent strategies. As a basis for this type of behavior, a flexible formation structure has been developed (Stone and Veloso 1997). In this structure, each agent has the knowledge required to fill any team role in any of several team formations. Furthermore, the agents are equipped with low-level communication protocols allowing for seamless formation adjustments and dynamic role adjustments. The agents are also equipped with several pre-defined multiagent, multi-step plans that can be instantiated in the appropriate situations. This flexible formation has been successfully implemented on real robots as well as in simulation (Stone and Veloso 1997). Several learning opportunities exist within this flexible formation structure, including when to switch formations and/or positions and how to refine the pre-defined plans based on the current state of the world.

The existing learned behaviors in the robotic soccer domain could naturally map onto spacecraft constellation control. A neural network could be trained to do low level control of behaviors, while a decision tree could allow goal-driven commanding in a symbolic form. Higher-level collaborative behaviors could then be used to control multi-craft interactions such as relative positioning.

Of course, machine learning should not be used for processes that are easily automated without it. Layered learning prescribes the use of learning when hand-coding is non-trivial. Thus, low-level spacecraft control may not need to be learned. On the other hand, past control algorithms have been developed only with significant time and effort. Machine learning could potentially reduce both the time and effort required to develop such algorithms for new spacecraft. Furthermore, the complexity of coordinating the control of several spacecraft suggests many multiagent learning opportunities at the higher levels of control.

Related Work

The layered learning approach is somewhat reminiscent of Brooks' Subsumption Architecture (Brooks 1986) which layers control modules, allowing high-level controllers to override lower-lever ones. Each control level is capable of controlling the robot on its own up to a specified level of functionality. Brooks implements his approach on real robots, building controllers for simple tasks such as avoiding collisions and wandering around.

Mataric brings the Subsumption Architecture to a multiagent learning domain, building controllers on

top of a set of learned *basis behaviors* (Mataric 1995). Mataric's basis behaviors are chosen to be necessary and sufficient for the learning task, while remaining as simple and robust as possible. Since Mataric's robots were to learn social behaviors such as flocking and foraging, they were equipped with basis behaviors such as the ability to follow each other and the ability to wander without running into obstacles.

While layered learning also makes use of multiple behavior layers, the tasks considered are much more complex: the agents must be able to generalize across situations, handle adversaries, and achieve complex goals. In order to move quickly to high-level behaviors, the commitment to have every layer be completely able to control the robot is abandoned. Instead, many situation-specific (but as general as possible) behaviors are produced which are then managed by higher-level behaviors. Nevertheless, the idea of building higher levels of functionality on top of lower levels is retained. It is in producing the situation-specific behaviors that machine learning techniques are used.

In many multiagent domains, there is the need or opportunity for teammates to assume different roles in a joint endeavor. As described above, a flexible teamwork structure has been developed. In related work, Tambe discusses a framework in which agents can take over the roles of other teammates in a helicopter-combat domain (Tambe 1996a). In the learning context, Prasad et al. have created design agents that can learn which role to fill (Prasad *et al.* 1996).

In addition to reasoning about roles of teammates, Tambe's combat agents can also reason about the roles that opponents are playing in team behaviors (Tambe 1996b). By recognizing an opponent's action as a part of a larger team action, an agent is able to more easily make sense of the individual opponent's behavior with the goal of being able to predict the opponent's future actions. Tambe's work enhances previous work that aims at having agents deduce other agents' intentions through observation (Huber and Durfee 1995).

Conclusion

Layered learning as well as its associated flexible team structure are proving themselves useful in the robotic soccer domain. Since many of the complexities of this domain match the complexities of several space-related applications, there is great promise of transferring the layered learning methodology to such applications. In particular, applying multiagent learning techniques to the goal of achieving spacecraft constellation autonomy is an exciting prospect, both because it has the potential to increase the range of possible missions and because it could significantly reduce mission budgets.

Future work includes identifying the precise levels of spacecraft behavior for which machine learning will be useful. Beginning at the lowest behavior-levels which are difficult to hand-code, learned layers can then be built in a bottom-up, hierarchical manner.

Acknowledgements

This research is conducted jointly with Dr. Manuela Veloso. Thanks to Dr. Steve Chien for his contributions and comments.

References

Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2:14–23, 1986.

Steve Chien. Distributed, adaptive, self-commanding technology for spacecraft autonomy, 1996. New Technology Document for the New Millennium Autonomy IPDT.

Marcus J. Huber and Edmund H. Durfee. Deciding when to commit to action during observation-based coordination. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 163–170, Menlo Park, California, June 1995. AAAI Press.

Hiroaki Kitano, Yasuo Kuniyoshi, Itsuki Noda, Minoru Asada, Hitoshi Matsubara, and Ei-Ichi Osawa. Robocup: A challenge problem for ai. *AI Magazine*, 18(1):73–85, Spring 1997.

Maja J. Mataric. Designing and understanding adaptive group behavior. *Adaptive Behavior*, 4(1), December 1995.

M V Nagendra Prasad, Victor R. Lesser, and Susan E. Lander. Learning organizational roles in a heterogeneous multi-agent system. In *Adaptation, Coevolution and Learning in Multiagent Systems: Papers from the 1996 AAAI Spring Symposium*, pages 72–77, Menlo Park,CA, March 1996. AAAI Press. AAAI Technical Report SS-96-01.

R. Ridenoure, June 1995. New Millenium Mission Operations Study.

Peter Stone and Manuela Veloso. Task decomposition and dynamic role assignment for real-time strategic teamwork. In *submitted to the Third International Conference on Multi-Agent Systems*, November 1997. Draft available by request or from http://www.cs.cmu.edu/ pstone/pstone-papers.html.

Peter Stone and Manuela Veloso. A layered approach to learning client behaviors in the robocup soccer server. *To appear in Applied Artificial Intelligence (AAI) Journal*, 1998.

Milind Tambe. Teamwork in real-world, dynamic environments. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*, Menlo Park, California, December 1996. AAAI Press.

Milind Tambe. Tracking dynamic team activity. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Menlo Park, California, 1996. AAAI Press.

A Unified Approach To Network Optimization of Satellite Communications Systems

Keiji Tasaki
Networks Division/Code 530
NASA/Goddard Space Flight Center
Greenbelt, MD 20771
keiji.tasaki@gsfc.nasa.gov

Joseph Nickey
Stanford Telecommunications, Inc.
7501 Forbes Blvd., Suite 105
Seabrook, MD 20706
jnickey@stel.gsfc.nasa.gov

Robert J. Hollingshead
Stanford Telecommunications, Inc.
7501 Forbes Blvd., Suite 105
Seabrook, MD 20706
rhollingshead@stel.gsfc.nasa.gov

Abstract

This paper describes a networks optimization approach that is being investigated by the Networks Division in response to the need to efficiently handle the increasing demands of users of the Space Network (SN), Deep Space Network (DSN), and Ground Network (GN) resources (antennas, receivers, etc.), while reducing the operational and developmental cost of all communication systems.

Introduction

The Space Operations and Management Office (SOMO) of the National Aeronautics and Space Administration (NASA) is engaged in strategic planning and system engineering activities to promote and enhance the services offered to current and potential customers of NASA networks. In order to facilitate these activities, the Networks Division at NASA's Goddard Space Flight Center has been conducting research into the development of a unified approach for modeling satellite communications with the goal of generating and assessing future network architectures.

Objective

The primary objective of this project is to develop an integrated PC-based tool capable of generating solutions to problems of the following type:

Given:

1. A set of network resource elements $\{E_1 \ldots E_n\}$ (representing SN, DSN, and GN antennas, receivers, etc.); and

2. A set of missions $\{M_1 \ldots M_i\}$

Determine the optimum communication strategy for mission M_{i+1} when it is added to the mission set.

We will refer to the problem of determining whether a new mission can be supported adequately by a given set of network resources as a Type I problem as depicted in Figure 1.

In addition, this tool will find application as an aid in solving long term strategic planning problems where one is interested in examining how changes in network elements affect supportability of a given mission set and the companion problem of how best to configure a set of network elements for optimal support of a given mission set. We will refer to problems of this class as Type II problems as depicted in Figure 1.

In summary, the objective of the Network Optimization project is to develop a process by which:

1. A series of technically feasible space-to-ground and ground-to-space communications networks are rapidly assessed for any given mission (Type I problem), and

2. An optimum network solution can be identified for a given mission set based on cost, contact time, and other critical factors (Type II problem).

In the following we provide a overview of each step in the optimization process followed by a brief description of our progress to date in implementing each segment of the system.

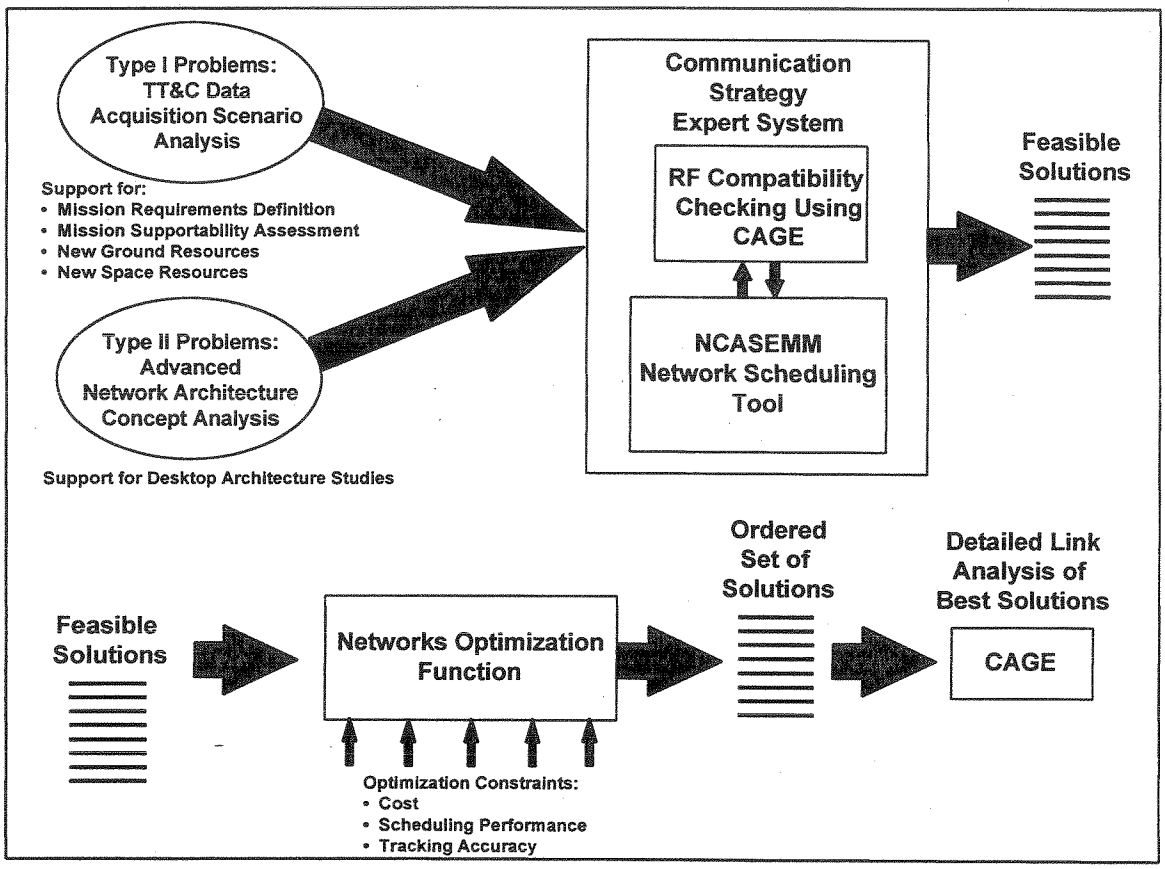


Figure 1. Network Optimization Applications.

A Five Step Approach to Optimization

Figure 2 depicts the system-level optimization process. As shown, the first step in the process is to develop an expert system front end to automate the task of converting mission requirements into a communication strategy in support of specific communication network service requests. The expert system will interrogate the user for mission communication requirements such as:

1. Orbital Information
   - Keplerian orbital elements
2. Downlink Requirements
   - Data volume per day (science)
   - Data volume per day (housekeeping)
   - Onboard storage capability (drives contact duration and data rate required)
   - Required notification of celestial events
3. Uplink Requirements
   - Data volume per day (commanding)
   - Stored command capability
4. Spacecraft Antenna Characteristics
   - Operating frequencies
   - Transmitter and Receiver characteristics
5. Control Center Location Information
   - Mission Operations location
   - Science Operations location

The expert system will then generate a set of preliminary communication alternatives that satisfy the requirements set forth by the user. A typical contact requirement for an Earth-observing mission with a 100-minute, polar orbit would be two 10-minute contacts per orbit at X-band (5.2 - 10.9 GHz) for science data downlink and three 5-minute contacts every two orbits for spacecraft housekeeping at S-band (1.5 - 5.2 GHz).

It is important to note that at this stage in the process the communication scenarios are only constrained by the mission requirements and the known *existence* of a particular communication service (e.g., TDRSS S-Band Single Access (SSA)). The actual *availability* of the service to a given mission at a specific instant is not a constraint at this stage. It is therefore likely that many of the communication scenarios that are generated at this stage are not going to be suitable final solutions because they may depend on the availability of finite resources that are in fact not available at the requested time due to preemption by other missions. The problem of assigning communication services among competing users in a nearly optimal fashion will be the subject of the next step in the optimization process where we consider the global optimization problem of allocating finite communication resources among competing missions.

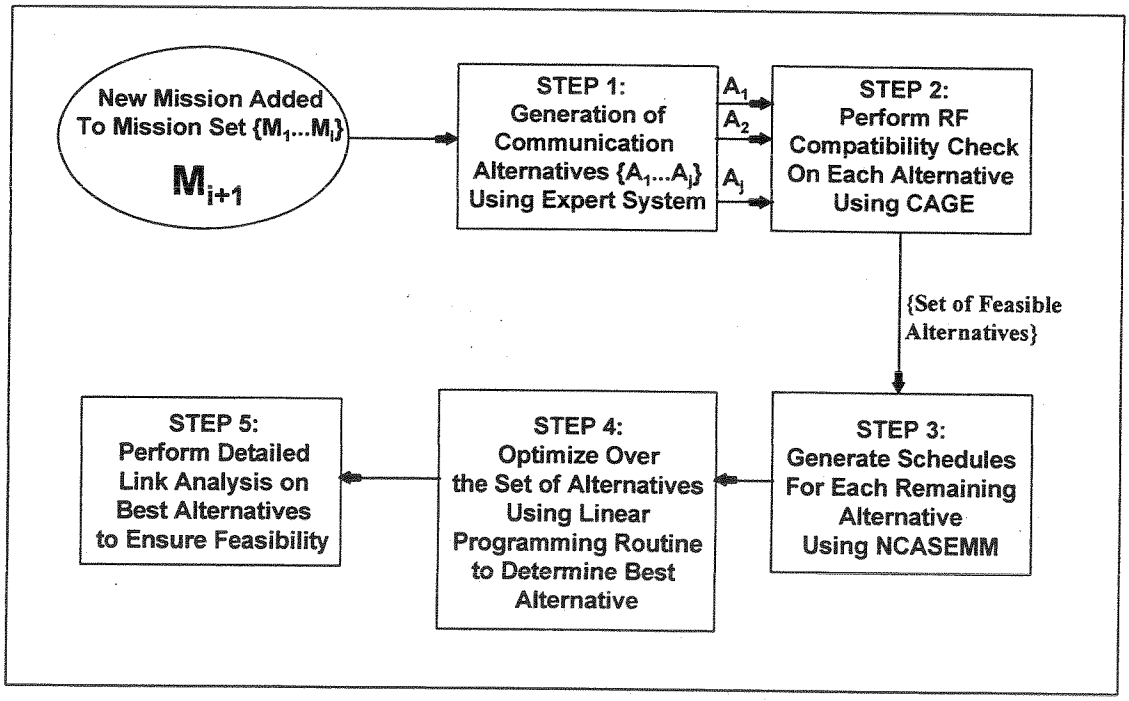


Figure 2. Network Optimization Process Flow.

The Second Step in the Optimization Process: RF Compatibility

The second step in the optimization process is to perform RF compatibility checking on the user specified contact and RF requirements to guard against attempting to schedule across incompatible or non-existent resources. We have identified a COTS product called the Configurable Analysis Graphical Environment for Space Systems (CAGE) that runs on a PC platform and can be adapted for this purpose. In addition to other useful features, CAGE allows easy modeling of link performance, interference and frequency assignments and it contains numerous libraries that model transmitters, receivers, antennas and other communication components. We are currently in the process of investigating the feasibility of interfacing it with the scheduling tool that we have selected which is discussed in the next section.

The Third Step in the Optimization Process: Generate Feasible Solutions

The third step in the optimization process is the generation of a set of feasible communication solutions that satisfy the requirements for a new mission (Type I problem) or a new network configuration (Type II problem). A feasible solution is defined as a complete network/mission set pair that is capable of supporting the requirements of a given mission set. A scheduling simulation tool will be used to determine supportability. This process is repeated a number of times with different network architecture or mission set scenarios to generate a series of alternatives each of which satisfies all communication requirements. Each alternative represents a complete communications network consisting of elements of tracking stations, transportable tracking terminals, tracking and data relay satellites, and/or commercial services.

The NASA Computer Aided System Engineering and Management Model (NCASEMM) was developed by Veda, Inc. as a PC-based scheduling support tool that generates network communication schedules for non-operational mission planning purposes. NCASEMM accepts as inputs the mission orbital characteristics, desired mission communication contact requirements in terms of event duration and frequency, and the communication resources that are available for scheduling such as ground based or space based (TDRSS) tracking stations. It then performs a scheduling function that attempts to best accommodate all requirements for all missions.

Figure 3 depicts the results of a typical NCASEMM run. Note the imperfect scheduling that results from finite availability of resources and other inherent limitations characteristic of the scheduling algorithm.

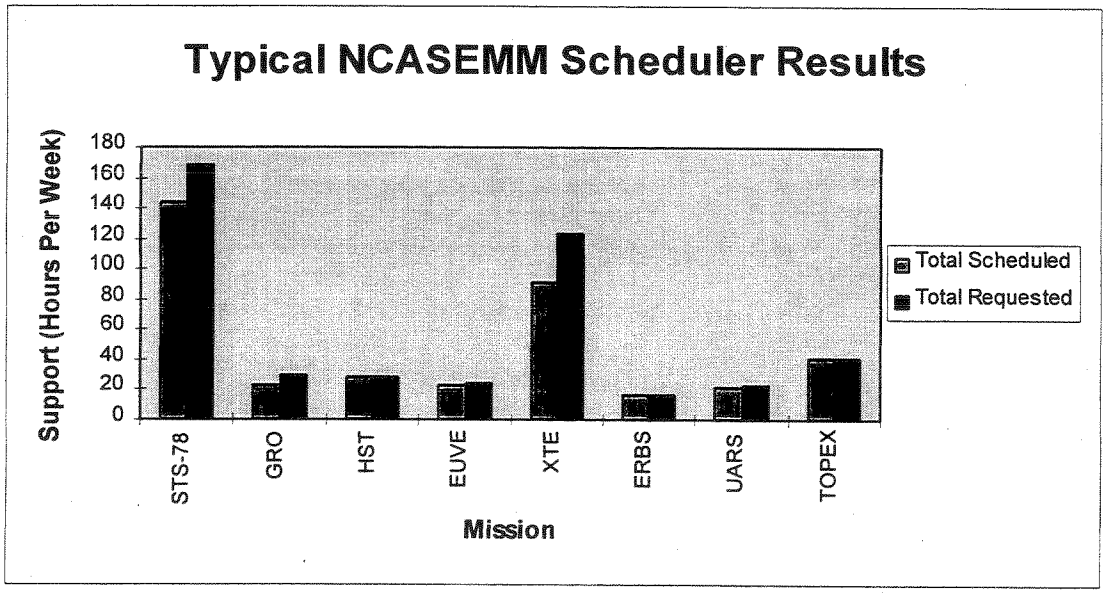


Figure 3. NCASEMM scheduler results for a NASA Space Network mission set.

We have performed an in-depth assessment of NCASEMM and found that the performance of the scheduler is strongly dependent on the manner in which the user specifies the contact requirements. We have concluded that with minor enhancements NCASEMM is capable of performing the scheduling function in our optimization plan.

The Fourth Step: Optimization of the Set of Feasible Solutions

At this stage in the process we have generated a set of technically feasible communication architecture solutions, all of which are capable of satisfying the requirements of the mission set under consideration. The fourth step in the optimization process is to optimize over the set of feasible solutions to identify the preferred solution. In order to perform this task, one must first identify relevant constraints that are to be considered. Our focus has been to minimize on the cost of communication.

The major factors contributing to cost include: service charge per minute for the use of the TDRSS, DSN, or GN; the fabrication and installation costs of transportable tracking terminals for dedicated use; the maintenance and operation of the terminals; service charges associated with the use of commercial providers; service charges associated with the use of tracking stations; and the cost of the onboard communications equipment (e.g., transmitters, receivers, transponders, antennas, etc.). These cost factors are added together to form an objective function for the cost minimization problem. The primary set of constraints, on the other hand, includes the total budget available for communications for the mission under study and the total service time available at each resource location. It is clear that when this minimization problem is kept at this level of complexity, it can be solved straightforwardly using a variety of methods including linear programming. However, when additional factors are taken into account, such as variable service pricing structures for priority users, operating hours of tracking facilities, and scheduling of hand-over from commercial services to a NASA-managed tracking station, the minimum cost problem becomes extremely complex. In addition, some of the constraint equations become non-linear. For these reasons, the current effort has concentrated on the simpler model, tabling the more challenging issues for the time being. This iterative process will produce an ordered set of communications alternatives which will be very useful for decision-makers who are responsible for planning the communications systems for future missions.

The Fifth Step: Detailed Link Analysis of Preferred Solutions

The fifth and final step in the optimization process is to perform a detailed link analysis on the preferred solution and any other highly optimal solutions obtained from step four. This will ensure that the optimal solutions are truly capable of performing as anticipated. We currently envision using CAGE to perform this function.

Summary

This Network Optimization project has been an on-going effort for the past twelve months. Resource allocation, linear programming, and decision theory are key to solving this complex problem of optimally developing and assigning limited and costly communications resources to satisfy the demands made by new missions. Although the project is still in the early stages, the scheduling scenarios that we have generated using the NCASEMM scheduling tool are promising. Upon completion of the Network Optimization project, it is anticipated that the outlined process will significantly aid in assessing and designing communications network architectures for future NASA missions.

A Reactive Planner for a Model-based Executive*

Brian C. Williams
Computational Sciences Division MS 269-2
NASA Ames Research Center,
Moffett Field, CA 94035 USA
E-mail: williams@ptolemy.arc.nasa.gov

P. Pandurang Nayak
Recom Technologies
NASA Ames Research Center, MS 269-2
Moffett Field, CA 94035 USA
E-mail: nayak@ptolemy.arc.nasa.gov

Abstract

A new generation of reactive, model-based executives are emerging that make extensive use of component-based declarative models to analyze anomalous situations and generate novel sequences for the internal control of complex autonomous systems. Burton, a generative, model-based planner offers a core element that bridges the gap between current and target states within the reactive loop. Burton is a sound, complete, reactive planner that generates a single control action of a valid plan in average case constant time, and compensates for anomalies at every step. Burton will not generate irreversible, potentially damaging sequences, except to effect repairs. We present model compilation, causal analysis, and online policy construction methods that are key to Burton's performance.

Conventional wisdom has largely pushed deductive reasoning out of the reactive control loop for nearly a decade. However, recent search for the surprisingly elusive, hard satisfiability problem foretells a healthy return to deductive methods(Williams & Nayak 1996b; Kautz & Selman 1996) based on RISC-like search engines. This paper pushes this perspective down to reactive time scales, reporting on a model-based planner, called *Burton*, that is at the core of a model-based executive's reactive control loop. By solving the NP hard component of deductive problems at compile time, Burton exploits the expressiveness of NP hard methods, without assuming the risk of falling off the elusive cliff.

Burton's parent model-based executive is particularly well suited to controlling the complex internal behaviors of large scale autonomous systems, we call *immobile robots* (Williams & Nayak 1996a). What distinguishes this executive is its ability to sense and control hidden state variables indirectly, and the use of component models to identify these novel interaction paths. A marriage between this model-based executive and a classical, method-based executive provides a hybrid with an expressive scripting language and an extensive capability to generate novel responses to anomalous situations. Significant parts of this hybrid executive will be demonstrated in late 1998 on NASA's Deep Space One autonomous spacecraft (Pell *et al.* 1997).

The paper begins with an example from the spacecraft domain, and then introduces our concurrent transition system modeling formalism. Next we introduce model-based execution as identifying a current state (mode identification), generating an optimal target state (mode reconfiguration), and generating a control action to move towards the target (model-based reactive planning). The rest of the paper presents the Burton model-based reactive planner through a series of domain restrictions, model compilation, policy construction and online planning algorithms.

Example: autonomous spacecraft

First consider the underlying task. Figure 1 shows an idealized schematic of the main engine subsystem of the Cassini spacecraft and valve driver circuitry. It consists of a helium tank, two propellant tanks, two main engines, regulators, latch valves, and pyro valves. The helium tank pressurizes the propellant tanks. When propellant paths to a main engine are open, the propellants flow into the engine and produce thrust. The pyro valves are used to isolate parts of the engine. They can open or close only once.

Valves are controlled by valve drivers. Commands to the driver are sent via a control unit (VDECU). The driver and VDECU can be on or off, and recoverably or permanently failed. A recoverably failed component can be repaired by resetting it. A valve's state changes as a result of a command only if the corresponding driver and VDECU are on and healthy.

In planning an orbit insertion maneuver, a high-level deliberative planner, *e.g.*, (Muscettola 1994), generates a sequence of behavior goals, such as producing thrust. The reactive executive achieves this goal using its component models to generate control sequences

* This paper appears in *Proceedings of IJCAI-97.*

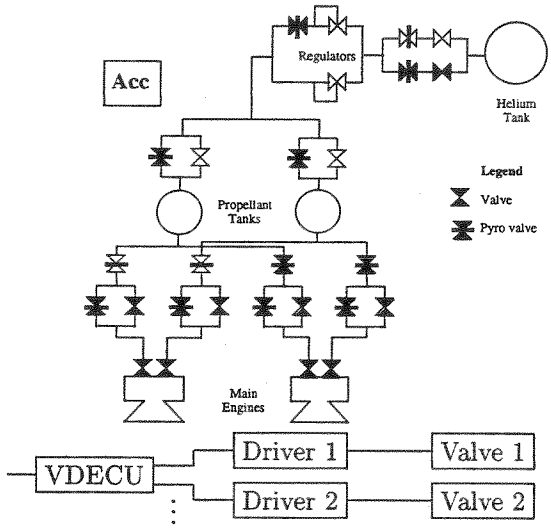


Figure 1: Schematics of engine and valve control circuitry. Valves are closed only when solid black.

that open the relevant set of valves leading to a main engine. Valves are commanded open indirectly, and the executive must ensure that the control unit and driver leading to the valve are on and healthy prior to commanding the valve. Generating sequences to handle a breadth of novel situations requires extensive reasoning about physical processes as well as state changing actions. Doing this reactively is the focus of this paper.

Concurrent transition systems

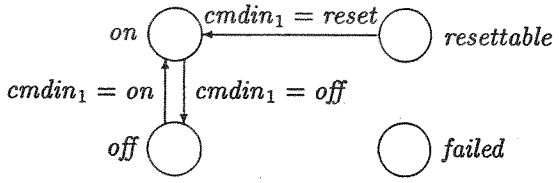
We start by reviewing the propositional, concurrent transition system formalism introduced in (Williams & Nayak 1996b), which represents normal operation, failure, and repair of real-time software and hardware. A *transition system* $\mathcal{S}$ is a tuple $\langle \Pi, \Sigma, \mathcal{T} \rangle$. Π is a set of *variables*, each ranging over a finite domain. Π is partitioned into the set Π_s of *state* variables, the set Π_c of *control* variables, and the set Π_d of *dependent* variables. Σ is the set of *feasible* assignments to variables in Π. Each element of a state variable's domain is categorized nominal or failure. Control variable values are determined exogenously by a controller. A *state* is an assignment to each variable in Π_s. The set Σ_s, the projection of Σ on variables in Π_s, is the set of all feasible states. $\mathcal{T}$ is a finite set of *transitions*. Each transition $\tau \in \mathcal{T}$ is a function $\tau : \Sigma \rightarrow \Sigma_s$, *i.e.*, $\tau(\sigma)$ is the state obtained by applying transition τ to any feasible assignment σ. The transition $\tau_n \in \mathcal{T}$ models the system's nominal behavior, while all other transitions model failures. A repair action is a transition that takes a state variable from a failure to nominal value.

A *trajectory* is a (finite or infinite) sequence of feasible states $S : s_0, s_1, \ldots$ such that for each s_i there is a feasible assignment $\sigma_i \in \Sigma$ which agrees with s_i on assignments to variables in Π_s and $s_{i+1} = \tau(\sigma_i)$ for some $\tau \in \mathcal{T}$. A trajectory that involves only the nominal transition τ_n is called a *nominal trajectory*. A *simple trajectory* does not repeat any state.

A transition system is specified using a restricted subset of propositional temporal logic that uses the $\bigcirc$ operator. The propositions in this logic are all of the form $y = e$, where y is a variable and e is an element in y's domain. $\bigcirc$ is the *next* operator denoting truth in the next state of a trajectory.

We specify the set Σ of feasible assignments by a propositional formula ρ_Σ, *i.e.*, Σ is the set of all variable assignments that satisfy ρ_Σ. A transition τ of $\mathcal{S}$ is specified by a formula ρ_τ, which is a conjunction of *transition specifications* ρ_{τ_i} of the form $\Phi_i \Rightarrow \bigcirc y_i = e_i$, where Φ_i is a propositional formula and y_i is a state variable. A state s_k can follow state s_j using τ if there is an assignment $\sigma_j \in \Sigma$ that agrees with s_j such that for all ρ_{τ_i} if σ_j satisfies Φ_i then s_k assigns e_i to y_i.

Example 1 Driver 1 has a state variable, dr_1, and two dependent variables $cmdin_1$ and $cmdout_1$. The domain of dr_1 is $\{on, off, resettable, failed\}$, $cmdin_1$ is $\{on, off, reset, open, close, none\}$, and $cmdout_1$ is $\{open, close, none\}$. dr_1's transition diagram is:



which is specified by formulae like:

$$dr_1 = resettable \wedge cmdin_1 = reset \Rightarrow \bigcirc dr_1 = on$$

which describes the effect of resetting a driver. The driver's feasible states are specified by formulae like:

$$(dr_1 = on \wedge cmdin_1 = open) \Rightarrow (cmdout_1 = open)$$

The VDECU has control input $drcmdin_1$, whose value is propagated to $cmdin_1$ when the VDECU is *on*:

$$(vdecu_1 = on \wedge drcmdin_1 = reset) \Rightarrow (cmdin_1 = reset)$$

Opening the valve is achieved by turning the VDECU and driver on, and then setting control input $drcmdin_1$ to *open*. The driver's output $cmdout_1$ is used to control the valve's state transitions.

Model-based reactive execution

We view an autonomous system as a high-level *planner* coupled to a low-level *reactive control system*. The

planner generates a sequence of configuration goals, each a physical behavior like "achieve thrust." The reactive control system evolves along a trajectory that achieves these configuration goals. Reactive control is achieved using a *reactive executive* that generates a sequence of control actions, based on knowledge of the current state and configuration goal. A control action is an assignment to each control variable in Π_c, for example, corresponding to closing a switch or sending a reset message across a bus. The current state is (partially) observable through a set of variables $\Pi_o \subseteq \Pi_s \cup \Pi_d$, corresponding to sensors, and an observation is an assignment to each variable in Π_o.

Definition 1 A reactive control system is a tuple $\langle \mathcal{S}, \Theta, \mathcal{C} \rangle$, where $\mathcal{S}$ is a transition system, Θ is the initial state of $\mathcal{S}$, and $\mathcal{C}$ is a reactive executive. $\mathcal{C}$ takes as input the initial state Θ, a sequence $\gamma : g_0, g_1, \ldots$ of propositional formulae called *goal configurations*, and a sequence of observations $o : o_0, o_1, \ldots$, and incrementally generates a sequence of control actions $\mu : \mu_0, \mu_1, \ldots$ so that $\mathcal{S}$ evolves along a trajectory $s : s_0, s_1, \ldots$ that satisfies: (a) s_0 is Θ; (b) for each s_i there is an assignment $\sigma_i \in \Sigma$ that agrees with s_i, o_i, and μ_i on the corresponding subsets of variables; and (c) if s_{i+1} is the result of a nominal transition from s_i under μ_i, *i.e.*, $s_{i+1} = \tau_n(\sigma_i)$, then either s_{i+1} satisfies g_i or $\langle s_i, s_{i+1} \rangle$ is the prefix of a simple nominal trajectory that ends in a state s_j that satisfies g_i.

The idea is that a reactive executive continually tries to transition the system toward a state that satisfies the desired goals. It is reactive in the sense that it reacts immediately to changes in goals and to failures, *i.e.*, each control action μ_i is incrementally generated using the new information, o_i and g_i, in each state.

A *model-based executive*, uses a specification of a transition system to determine the desired control sequence in three stages—*mode identification* (MI), *mode reconfiguration* (MR) and *model-based reactive planning* (MRP). MI and MR set up the planning problem, identifying initial and target states, while MRP reactively generates a plan solution. More specifically, MI incrementally generates the set of most likely plant trajectories consistent with the plant transition model and the sequence of observations and control actions. This is maintained as a set of most likely current states. MR uses a plant transition model and the most likely current state generated by MI to determine a reachable target state that satisfies the goal configuration. MRP then generates the first action in a control sequence for moving from the most likely current state to the target state. After that action is performed MI confirms that the intended next state is achieved. MI and MR are discussed in (Williams & Nayak 1996b). This paper focuses on MRP.

A key decision underlying our model-based executive is the focus on the most likely trajectory generated by MI. The difficulty with the more conservative strategy of considering a single control sequence that covers a set of likely states (Williams & Nayak 1996b) is that the different states will frequently require different control sequences. While utility theory can be used to select between different control sequences for one that maximizes success (Friedrich & Nejdl 1992), the cost of generating multiple states and control sequences works against our goal of building a fast reactive executive.

The greedy approach introduces risk: the control action appropriate for the most likely trajectory may be inappropriate, or worse still damaging, if the actual state were otherwise. Furthermore, the reactive focus of the MRP precludes extensive deliberation on the long-term consequences of actions, thus leaving open the possibility that control actions, while not outright harmful, may degrade the system's capabilities. For example, firing a pyro valve is an irreversible action that forever cuts off access to parts of the propulsion system. Such actions should be taken after due deliberation by the high-level planner or human operators. Hence, fundamental to the reliability of our approach is the following requirement:

Requirement 1 MRP considers only reversible control actions, unless the only effect is to repair failures.[1]

Model-based reactive planning

The definition of MRP (Burton) follows from Definition 1 and the functions of MI and MR:

Definition 2 A *model-based reactive planner* (MRP) takes as input a specification of a transition system $\mathcal{S}$, a most likely initial state s_i (from MI), and a lowest cost target state t_i (from MR) that satisfies goal g_i. The MRP generates a control action μ_i such that for any assignment $\sigma_i \in \Sigma$ that agrees with s_i and μ_i, the state $s_{i+1} = \tau_n(\sigma_i)$ is either the target state t_i or $\langle s_i, s_{i+1} \rangle$ is the prefix of a simple nominal trajectory that ends in t_i.

Given the similarity of transitions and STRIPS operators, Burton's problem appears similar to STRIPS planning (Weld 1994). However, the critical difference is the distinction between classical planning and machine control. The primitive control action in a STRIPS plan is to invoke a plan operator, which *directly* modifies the state. Model-based planners, on the other hand, exert control by establishing values for control variables, which interact with internal state

[1] While repairing a failure is irreversible, it is important to allow a reactive executive to repair failures.

variables *indirectly* through co-temporal, physical interactions (ρ_Σ). This extension adds enormous expressivity to the planner.

This extension equally adds to the challenge of achieving reactivity. We eliminate intractability at modeling time through an automated model compilation method and four simple modeling requirements.

Compiling transition models

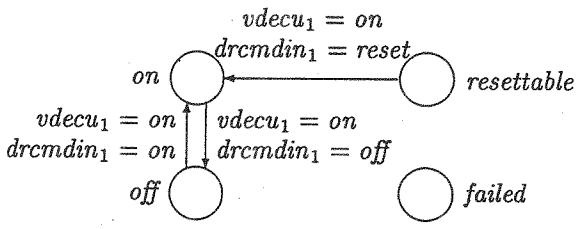
Recall that a transition is specified by a conjunction of formulae $\Phi_i \Rightarrow \bigcirc y_i = e_i$. What distinguishes this from a simple transition is that the formula Φ_i can contain propositions involving dependent variables that are not directly controllable, but depend on control variables through the arbitrary propositional formulae of ρ_Σ. This introduces a potential computational cliff.

To simplify the transition system, we first *compile away* all dependent variables and corresponding interactions of ρ_Σ, by generating all *prime implicants* of $\bigcirc y_i = e_i$ that do not contain dependent variables. More precisely, an implicant of $\bigcirc y_i = e_i$ is a conjunction I of propositions involving state and control variables such that the transition system specification, $\rho_\Sigma \cup \bigcup_{\tau \in \mathcal{T}} \rho_\tau$, entails the formula $I \Rightarrow \bigcirc y_i = e_i$. I is a prime implicant if no sub-conjunction of I is an implicant. Given that dependent variables cannot be directly controlled, either by control actions or by transitions, the prime implicants of $\bigcirc y_i = e_i$ completely characterize the conditions under which y_i has the value e_i in the next state. For example, one prime implicant of $\bigcirc dr_1 = on$ (from Example 1) is

$$dr_1 = resettable \wedge vdecu_1 = on \wedge drcmdin_1 = reset$$

While prime implicant generation is NP-hard, in practice even sizable implicants can be generated very fast. For example, we use an abductive best-first search (Williams & Nayak 1996b) to generate implicants from a spacecraft model consisting of over 12,000 clauses in about 40 seconds on a Sparc 20. We compute these implicants at compile-time, thus avoiding the risk of falling over the computational cliff at run-time, while preserving expressivity in the modeling language.

The set of prime implicants constitutes the compiled transition specification, where each implicant specifies a transition for a single state variable. The transition specification of a single variable y_i is the set of prime implicants of $\bigcirc y_i = e_{ik}$, for every e_{ik} in y_i's domain. Antecedents of transition specifications now contain only state and control variables. For a transition of y_i, each antecedent involving a state variable other than y_i is called a *state condition*, and each antecedent involving a control variable is called a *control condition*. If y_i is not in the antecedent, we replicate the transition for every assignment of y_i. The transition diagram for the driver after compilation is:



The correspondence to a STRIPS operator is straightforward: state conditions, including antecedent y_i, form preconditions; the add (delete) list contains the next (previous) value of y_i. The transition is invoked by asserting all control conditions.

Exploiting properties of designed systems

Albeit simpler than the original specification, these compiled transitions have a complication not found in STRIPS. STRIPS operators are explicitly invoked, one at a time. In the above, a control action can invoke more than one transition. Furthermore, a transition can occur spontaneously, and cannot be prevented, if its antecedent contains no control variables.

While arbitrary transition systems can manifest these properties, physical hardware is typically designed to behave like STRIPS operators. It is usually the case that each state variable is separately commanded and state variables maintain their values in the absence of explicit commands. The following requirement prevents spontaneous state change:

Requirement 2 Each control variable has an idling assignment, and no idling assignment appears in any transition. The antecedent of every transition includes a (non-idling) control condition.

For example, $drcmdin_1$ has idling value *none*, and the prime implicant of $\bigcirc dr_1 = on$ mentioned above contains the non-idling $drcmdin_1 = reset$. Exploiting this restriction, all state changes can be prevented by assigning every control variable its idling assignment. The following additional restriction guarantees that transitions can be individually invoked:

Requirement 3 No set of control conditions of one transition is a proper subset of the control conditions of a different transition.

A single transition is invoked by asserting its control conditions, and assigning all other control variables their idle assignment. Jointly these two requirements reduce MRP to STRIPS planning.

Online component of reactive planning

Given a compiled transition system S', Burton quickly generates the first control action of a valid plan, given

an initial state assignment Θ and a set of goal assignments γ.[2] Burton meets five desiderata. First, it only generates *non-destructive actions*, *i.e.*, an action will never undo the effects of previous actions that are still needed to achieve top-level goals. Second, Burton will not propose actions that lead to *deadend plans*, *i.e.*, it will not propose an action to achieve a subgoal when one of the sibling subgoals is unachievable. Third, Burton is *complete*, *i.e.*, if a planning problem that satisfies Requirements 1-4 is solvable, then Burton will generate a plan. However, Burton is not guaranteed to generate all valid plans. Fourth, Burton ensures *progress* to a goal, except when execution anomalies interfere, *i.e.*, the nominal trajectory traversed by Burton for a fixed target is loop free. Fifth, Burton operates at *reactive time scales*—its average runtime complexity is constant. This speed is essential to providing a model-based executive with response times comparable to traditional executives (Firby 1987; Simmons 1994).

Burton avoids runtime search, requires no algorithms for threat detection, and expends no effort determining future actions or planning for subgoals that are not supported by the first action. Traditional planners need such mechanisms to avoid destructive actions and deadend plans (Weld 1994). Burton accomplishes this speedup by exploiting the requirement, stated earlier, that all actions except repairs be reversible, and by exploiting certain topological properties of component connectivity that frequently occur in designed systems. The development of Burton's basic sequencing algorithm is the topic of the next three subsections, with the introduction of repair actions introduced in the fourth subsection. Finally, to achieve average case constant time Burton precompiles plans into reactive policies, without requiring enormous amounts of storage. This is the topic of the fifth subsection.

Exploiting causality

A major leverage point comes from exploiting the topological properties of component connectivity. The input/output connections of the compiled plant frequently do not contain feedback loops. When they do occur they are typically local and can easily be eliminated through careful modeling. To be precise:

Definition 3 A *causal graph* $\mathcal{G}$ for (compiled) transition system $\mathcal{S}$ is a directed graph whose vertices are the state variables of $\mathcal{S}$. $\mathcal{G}$ contains an edge from $v1$ to $v2$ if $v1$ occurs in the antecedent of one of $v2$'s transitions.

Requirement 4 The causal graph must be *acyclic*.

[2]Here on, each goal is a target variable assignment generated by MR, not the configuration goal input to MR.

The causal graph for the valve circuitry follows the schematic, and is acyclic.

The basic idea underlying Burton is to solve a conjunction of goals by working "upstream" along the acyclic causal graph, *i.e.*, it completes a conjunct $y_i = e_{ik}$ before conjunct $y_j = e_{jk}$ when y_j precedes y_i in the causal graph. For example, suppose the initial state has $vlv_1 = closed \wedge dr_1 = off$, and the target has $vlv_1 = open \wedge dr_1 = off$. Burton first focuses on $vlv_1 = open$ since the valve is downstream of the driver. As we will see, this upstream progression guarantees that Burton generates non-destructive actions without an explicit threat detection mechanism.

Burton generates a next control action with the NextAction function, shown below. This takes an initial state Θ, a (partial) target state γ (sorted by topological number as discussed later), a compiled transition system S' and the flag **top?** to indicate a top-level call. NextAction returns a next control action, **Failure** if no plan exists, or **Success** if the initial state is a target state. A control action is a partial set of control assignments. All control variables not mentioned are assigned their idle value.

Online Algorithm: NextAction(Θ,γ,S',**top?**)

1. **Solvable Goals?:** When **top?** = **True**, unless each goal $g \in \gamma$ is labeled **Reversible**, return **Failure**.
2. **Select unachieved goal:** Find an unachieved goal assignment with the lowest topological number. Goal $y = e_f \in \gamma$ is unachieved if it differs from y's initial assignment $y = e_i \in \Theta$. If all achieved return **Success**.
3. **Select next transition:** Let t_y be the transition graph in S for goal variable y. Find a path p in t_y from e_i to e_f along transitions labeled **Allowed**. Let SC and CC be the state and control conditions of the first transition along p.
4. **Enable transition:**
   Control = NextAction(Θ,SC,S',False). If Control = **Success** then state conditions SC are already satisfied, return CC to effect transition. Otherwise Control contains control assignments to progress on SC. Return Control.

Line 1 tests whether or not a conjunction of top-level goals can be achieved, as explained in the subsection after next. This involves a simple table lookup to see if each goal is labeled **Reversible**. Burton only introduces subgoals that can be achieved, hence the test is needed at the top level only. Line 2 works upstream along the causal graph of variables selecting the next unsatisfied (sub)goal assignment to be achieved. This upstream progression is achieved by exploiting a topological ordering, as explained in the next subsection.

Line 3 takes the first step towards achieving the selected goal. Given an initial assignment $y = e_i$ and a component transition system for y, a goal assignment $y = e_f$ is achieved by traversing a path along transitions of y from e_i to e_f. Respecting Restriction 1, Burton only traverses transitions with reversible effects or that perform a repair. As explained in the next section, the transitions that satisfy this restriction are those labeled **Allowed**. Line 3 identifies the first transition along this path from e_i to e_f. Line 4 subgoals on the state conditions of this first transition, which results in Burton moving further upstream. If one or more state conditions is unsatisfied then a next control action is computed in Line 4, and returned. If all state conditions are satisfied, then the transition is ready to be traversed and Line 4 returns the transition's control conditions as the next control action.

Avoiding destructive actions

Burton avoids generating destructive control actions (desiderata 1) by exploiting the acyclic nature of the causal graph. The only variables needed to achieve an assignment $y = e$ are y's ancestors in the graph. For example, turning on the driver requires use of the VDECU but not the valve. In addition, requirements 2 and 3 guarantee that invoking transitions for y and its ancestors, when performed one at a time, will not affect any other state variables.

This suggests that Burton can achieve a conjunction of goal assignments in an order that moves along the causal graph from descendants to ancestors, *i.e.*, a goal conjunct is achieved only after conjuncts that are its descendants. For example, $dr_1 = \mathit{off} \wedge vlv_1 = \mathit{open}$ is achieved by working on $vlv_1 = \mathit{open}$ then $dr_1 = \mathit{off}$. The same ordering holds for conjunctive subgoals, that is, state conditions of required transitions.

Destructive subgoal interaction may occur when a variable appears upstream as a subgoal to two conjuncts. To avoid this danger subgoals are achieved in depth first order, achieving one conjunct before starting on a second. For example, $dr_1 = \mathit{off} \wedge vlv_1 = \mathit{open}$ is achieved by turning on the driver, opening the valve, and finally turning the driver off. Achieving subgoals in depth first order also ensures that Burton always makes progress towards the goal (desiderata 4).

Depth first goal progression, together with the goal ordering constraint, is sufficient to ensure non-destructive actions. Depth first progression is imposed by Line 4 of NextAction. Line 2 imposes the ordering constraint without runtime cost through *topological numbering*. At compile time each state variable is given a topological number (tn), by performing a depth first search of the graph and numbering variables on the way out. For example, $tn(vlv_1) = 1, tn(dr_1) = 2, tn(vlv_2) = 3, tn(dr_2) = 4$, and $tn(vdecu_1) = 5$. Topological numbering imposes a total ordering that satisfies the constraint $tn(A) < tn(B)$ whenever A is a strict descendant of B in the causal graph. Hence the proper order of goal achievement for all conjunctive subgoals is determined at compile time by sorting the conditions of each transition by increasing topological number. Burton respects an upstream progression in line 2, simply by working successively through the sorted list of conditions.

Avoiding deadends through reversibility

Requirement 1 stated that Burton only perform reversible transitions, desiderata 2 stated that Burton avoid deadend plans, that is plans with unachievable subgoals, and desiderata 3 specified completeness. To be reactive Burton must achieve these without search. Each hinges on the following lemma:

Lemma 1 *$A \wedge B$ is reachable from Θ by reversible transitions exactly when A and B are separately reachable from Θ by reversible transitions.*

Proof: Assume without loss of generality that $tn(A) < tn(B)$. We previously showed that if A is achieved first, it won't be disturbed while achieving B. The transitions used to achieve A are reversible, hence *if nothing else* each variable other than A can be restored to its value in Θ and then B can be achieved. □

Suppose Burton has labeled as **Reversible** every assignment that is reachable from initial state Θ using **Reversible** transitions. By Lemma 1 a conjunction of goal assignments is achievable exactly when each conjunct is marked **Reversible**. Hence Burton can determine plan achievement at runtime simply by using one table lookup per top-level goal (line 1, NextAction).

Deadend plans, hence search, are eliminated by removing the possibility of unachievable subgoals. Assume Burton labels a transition **Allowed** only if its state conditions are each labeled **Reversible**. Then sequences of **Allowed** transitions between **Reversible** assignments contain no unachievable subgoals. These are the transitions generated by line 3 of NextAction. By lemma 1 all other sequences lead to deadends or involve irreversible transitions, hence Burton generates some plan if one exists (desiderata 3). Of course Burton can't generate all plans since subgoal interleaving isn't allowed. Finally, since depth first progression generates the control sequence in order, Burton generates the first control action without wasting work on the rest of the plan, hence achieving reactivity.

The labeling of assignments and transitions is precomputed for compiled transition system S' with initial state Θ by LabelSystem, and is linear in the size of S'.

Preprocess Algorithm: LabelSystem(S',Θ)
For each state variable y of S' in decreasing topological order:

1. For each transition τ_y of y, label τ_y **Allowed** if all its state conditions are labeled **Reversible**.
2. Compute the strongly connected components (SCCs) of the **Allowed** transitions of y.
3. Find y's initial value $y = e_i \in \Theta$. Label each assignment in the SCC of $y = e_i$ as **Reversible**.

LabelSystem starts at the roots of the causal graph, using topological numbering to move to a descendant only after its ancestors have been processed. Line 1 simply executes the definition of **Allowed**. Note when y_i is a root, none of its transitions contain state conditions, and hence are trivially **Allowed**. Lines 2 and 3 identify **Reversible** assignments. An assignment $y = e_k$ can be reversibly achieved if there exists a path along **Allowed** transitions from initial value e_i to e_k and back. Equivalently, the set of y's reversible assignments is the strongly connected component (SCC) of y's **Allowed** transition that contains e_i.

For example, starting with initial value *off*, the SCC for the VDECU, hence its **Reversible** assignment set, is $\{off, on\}$, with both *resettable* and *failed* excluded. Next, the driver has **Allowed** transitions between *on* and *off* due to the VDECU's SCC. With initial value *on*, the SCC for the driver is $\{off, on\}$.

Finally, note that LabelSystem is called infrequently. Actions generated by Burton only move within the SCCs of **Reversible** assignments, leaving the labeling unchanged. A relabeling is needed only if an exogenous action or failure moves Θ to an assignment that was not labeled **Reversible**.

Failure states and repair

To dramatically expand Burton's utility we incorporate repair actions. The occurrence of failures are outside Burton's control, since there are no nominal transitions that lead to failure. Hence a repair sequence is irreversible, albeit essential, and thus not covered by the development thus far. We extend Burton to permit repair sequences if they minimize irreversible effects. Burton never uses a failure to achieve a goal assignment if the failure is repairable. However, if it is not repairable, then Burton is allowed to exploit the component's faulty state. For example, suppose a switch is needed to be open, and it is permanently stuck open. Since stuck-open is irrepairable but has the desired effect, Burton exploits the failure mode.

Relaxing the reversibility constraint, if a state variable y is assigned a failure e_f, then Burton is permitted to traverse a sequence of allowed transitions from e_f to a nominal assignment, when such a path exists. If only one path exists, then y's reversible assignments are defined to be those in the SCC of the *first nominal assignment* reached along the path. For example if the driver is initially at *resettable* then it may transition to *on* using *reset*, and the SCC is $\{off, on\}$. If no path exists to a nominal assignment, then the reversible assignments are those in the SCC that contains e_f. For example, if the driver is at *failed*, then the SCC is simply $\{failed\}$. Although not discussed here, Burton also handles the case where multiple paths to different SCCs exist.

Although Burton can now traverse irreversible transitions to effect repair, none of the assignments along this trajectory, up to the selected SCC can be used to satisfy a state condition or goal assignment. Hence this extension does not endanger Burton's previously discussed properties.

Generating concurrent policies

The runtime complexity of NextAction is thus far $O(e*m)$, where e is the maximum number of transitions in a single component transition system, and m is the maximum depth of the causal graph. The cost m is incurred on line 4, while subgoaling on assignments of upstream variables, and cost e is incurred on line 3, while searching for a path through transitions. Burton reduces cost to $O(m)$ by constructing for each variable y a *feasible policy* π_y. π_y maps pairs $\langle e_i, e_f \rangle$ to the sorted conditions of the first transition along a path from e_i to e_f. If y has n values, π_y is simply an n^2 table computed in $O(n * e)$ time, where entries for each e_i are determined by computing a spanning tree directed towards e_f that connects all values labeled **Reversible** by transitions labeled **Allowed**. Line 3 of NextAction becomes the table lookup:

$$3' : \langle SC, CC \rangle = \pi_y(e_i, e_f)$$

vlv_1	Target	
Current	*open*	*closed*
open	**Idle**	$dr_1 = on$ $drcmdin_1 = close$
closed	$dr_1 = on$, $drcmdin_1 = open$	**Idle**
stuck	**Failure**	**Failure**

dr_1	Target	
Current	*on*	*off*
on	**Idle**	$drcmdin_1 = off$
off	$drcmdin_1 = on$	**Idle**
resettable	$drcmdin_1 = reset$	$drcmdin_1 = reset$

The driver and valve policies are shown above ($vdecu_1 = on$ is assumed and not shown). Suppose MI identifies the initial state $\{dr_1 = off, vlv_1 = closed\}$

and MR provides target $\{vlv_1 = open, dr_1 = off\}$, in topological order. Selecting the first target assignment, Burton looks up $vlv_1 : closed \rightarrow open$, and gets $\{dr_1 = on, drcmdin_1 = open\}$. The first is an unsatisfied state condition, so Burton looks up $\{dr_1 : off \rightarrow on\}$, getting $\{drcmdin_1 = on\}$. This control assignment is then invoked.

Burton's feasible policies are analogous to optimal policies in control theory. An important difference is that Burton constructs a set of concurrent policies, rather than a single policy for the complete state space. The latter grows exponential in the number of state variables, and is infeasible for models like the spacecraft, which contain over 80 state variables and well over 2^{80} states. In contrast, the concurrent policies grow only linearly in the number of state variables.

For a fixed target and no intervening failures, Burton generates successive control actions as a depth first traversal through the policy tables. This traversal maps out a subgoal tree, and Burton maintains an index into this tree during successive calls. To generate a sequence Burton traverses each tree edge exactly twice and generates one control action per vertex. Since the number of edges in a tree is bounded by the number of vertices, the *amortized average case complexity of generating each control action is a constant.* Desiderata 5, reactivity, is achieved.

Related work and conclusions

Burton combines causal models used in model-based reasoning with state transitions used in planning. Other researchers have combined model-based diagnosis with planning, primarily to generate repair plans (Friedrich & Nejdl 1992; Sun & Weld 1993). The main difference is that these systems include a STRIPS (rather than model-based) planning component with utility-theoretic measures for selecting amongst alternate plans. Their computational complexity make them inapplicable for on-board reactive execution.

Burton differs from traditional STRIPS planners (Weld 1994) in that plan operators (transitions) are generated by a compilation process from an underlying causal model of the system that includes both within and across state constraints (ρ_Σ and the ρ_τ, respectively). Furthermore, the compiled transition system is a specialized version of the general planning problem that Burton solves by a worst-case linear time, average case constant time algorithm. In contrast, general planning is PSPACE-complete. Korf (1987) defines a set of subgoals to be *serializable* if they can be solved in order without ever violating a subgoal solved earlier in the order. The requirement that the causal graph is acyclic ensures that Burton is only presented serializable sets of subgoals.

Finally, traditional reactive executives (Firby 1987; Simmons 1994) differ from Burton's model-based executive in that the former use explicitly scripted procedures to provide reactive execution, while the latter uses deductive reasoning from a causal model that combines an off-line compilation phase with an on-line policy generation phase. Other differences include the fact that traditional executives provide a richer set of control structures such as parallel execution, while Burton's executive provides a more sophisticated diagnosis and monitoring capability embodied in MI.

In summary, Burton is a sound and complete generative planner that uses expressive transition system models to provide the reactive sequencing capability of a model-based executive. It exploits off-line model compilation and the topological properties of component connectivity to incrementally generate control actions in average case constant time. These control actions are guaranteed to be non-destructive, while ensuring progress towards the goal and avoiding deadend plans.

Acknowledgements: We would like to thank Jim Kurien and Dan Weld for helpful comments on the paper.

References

Firby, R. 1987. An investigation into reactive planning in complex domains. In *Procs. of AAAI-87*, 202–206.

Friedrich, G., and Nejdl, W. 1992. Choosing observations and actions in model-based diagnosis/repair systems. In *Procs. of KR-92*, 489–498.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Procs. of AAAI-96*, 1194–1201.

Korf, R. 1987. Planning as search: A quantitative approach. *Artificial Intelligence* 33(1):65–88.

Muscettola, N. 1994. HSTS: Integrating planning and scheduling. In Fox, M., and Zweben, M., eds., *Intelligent Scheduling*. Morgan Kaufmann.

Pell, B.; Bernard, D.; Chien, S.; Gat, E.; Muscettola, N.; Nayak, P.; Wagner, M.; and Williams, B. 1997. An autonomous spacecraft agent prototype. In *Procs. of the First Int. Conf. on Autonomous Agents.*

Simmons, R. 1994. Structured control for autonomous robots. *IEEE Trans. on Robotics and Automation* 10(1).

Sun, Y., and Weld, D. 1993. A framework for model-based repair. In *Procs. of AAAI-93*, 182–187.

Weld, D. 1994. An introduction to least commitment planning. *AI Magazine* 15(4):27–61.

Williams, B., and Nayak, P. 1996a. Immobile robots: AI in the new millennium. *AI Magazine* 17(3):16–35.

Williams, B., and Nayak, P. 1996b. A model-based approach to reactive self-configuring systems. In *Procs. of AAAI-96*, 971–978.

Deterministic Scheduling In A Distributed Computing Environment

Song Yom, Karin Loya
Hughes Information Technology Corporation
1616 McCormick Drive
Upper Marlboro, MD 20774
{syom,kloya}@eos.hitc.com

Joe Hunt
Science Applications International Corporation
8301 Greensboro Drive
McLean, VA 22102
jdh@cip.saic.com

Abstract

A key application within NASA's Earth Observing System Data and Information System (EOSDIS) Core System (ECS) is the Planning and Data Processing System (PDPS) which enables fully-automated production processing of science data in a distributed, heterogeneous, multiprocessor, parallel computing environment. This paper describes the implementation of the planning and scheduling algorithm to generate efficient, near-optimal production schedules based on predicted data availability, site configuration, hardware/software resource availability, and on task load.

Introduction

NASA's Earth Observing System Data and Information System (EOSDIS) developed under the Mission to Planet Earth (MTPE) program is an open, distributed data and information system that will manage data from pre-EOS and EOS-era Earth observation satellites and other field measurement programs. EOSDIS will provide EOS instrument data collection, science data processing, and services for distribution of data holdings. The EOSDIS Core System (ECS) is the infrastructure of EOSDIS and includes the Planning and Data Processing System (PDPS) within which the science processing software is integrated.

The PDPS is comprised of three components: the first provides capabilities to support the integration of delivered science software into the production environment; second, the Planning Workbench, is the application framework for the planning and scheduling algorithm; and third, the science processing software execution manager which, with an embedded commercial-off-the-shelf (COTS) product, is the production engine.

The focus of this paper is the planning and scheduling algorithm, designed to:

- generate a near-optimal plan
- facilitate efficient use of resources in production
- enable forecasting of product availability
- allow analysis of alternative strategies to expedite product generation
- provide the base schedule to drive production over the operational window

The science software is profiled during the integration process, so that characteristics of the tasks to be planned have been pre-determined in advance of the planning and scheduling activity that is invoked through the Planning Workbench.

The Planning Workbench, implemented on an off-the-shelf C++ scheduling framework, performs deterministic scheduling based on the Highest Level First (HLF) List Scheduling algorithm otherwise known as the Critical Path Method (CPM). The task model consists of having precedence relationships between tasks enforced by data dependencies; non-preemptive, processor (CPU) allocation of task based on requisite runtime (size); communication delay embedded in task size; user definable relative priority for tasks; and earliest available start time for tasks based on external dependencies and forecast data arrivals. The resource model includes computers, processors, near-line/off-line storage system, and planned unavailability of these resources. No cache management of disks is considered by the algorithm. Internally, the system employs a linked-list implementation of a graph data structure for tasks; recursive graph traversal for calculating relative priority; and merge sort algorithm for task sorting based on user specified earliest start time, user specified priority, optimal start time, and calculated priority for tasks. The algorithm implements best-fit scheduling optimization by performing "look-ahead" and "look-behind" operations on an interval based multi-level linked-list resource data structure.

Problem Domain

The primary focus of the PDPS is the scheduling and execution of science processing software to automate the generation and archiving of data products. Output data product from an instance of a science processing software

(task) can be both an end product to be disseminated to the science community for analysis and an intermediate product to be used as input to another task.

The system is data- and event-driven. Request is made of the system to execute a task based on availability of required input data products which can be internally generated by another task and/or provided by an external data source such as raw spacecraft instrument data. As illustrated by Figure 1, tasks have precedence relationship enforced by data dependency. A task that depends on data produced by an upstream (predecessor) task must be scheduled after the upstream task. In addition, a task that is waiting on data from an external source must be delayed as necessary for data arrival. The task model also accounts for multiple tasks sharing one or more input data.

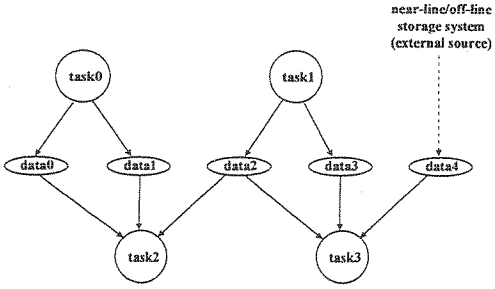


Figure 1. The Task Model describes the precedence relationship between tasks enforced by data dependency. A task can also depend on data arriving from an external source.

Tasks must be planned and scheduled on resources that consist of multiprocessor computing hardware, disks (local and network), and near-line/off-line storage system (dataserver). Dataserver is a hardware/software component of the ECS that provides data archiving and distribution services. For example, raw spacecraft instrument data ingested to the dataserver are retrieved by the PDPS as input to science processing software, and generated output data products are inserted back into the dataserver system for downstream processing.

The system must support planned resource unavailability. Resource reservations (also referred to in the ECS as ground events) can be made against processors, computers, and disks. Resources may be reserved for maintenance purposes. Forecast data arrival times can also be affected by scheduling ground events on the dataserver system. Load balancing is another factor for which the system must account. Over and under utilization of resources are to be avoided for efficiency of operation. Figure 2 describes the resource model.

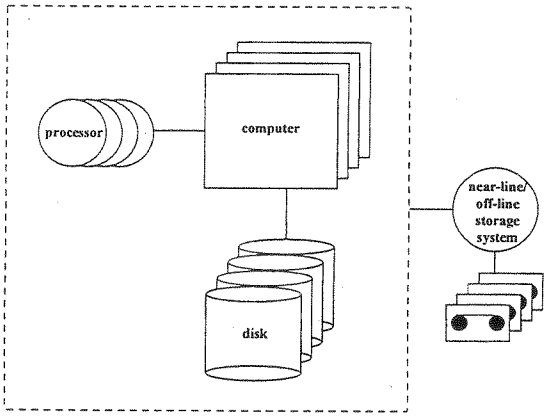


Figure 2. The Resource Model consists of multiprocessor computers with disks and a near-line/off-line storage system. It also includes planned unavailability of resources.

Merging the two models yields the basic problem domain view shown in Figure 3.

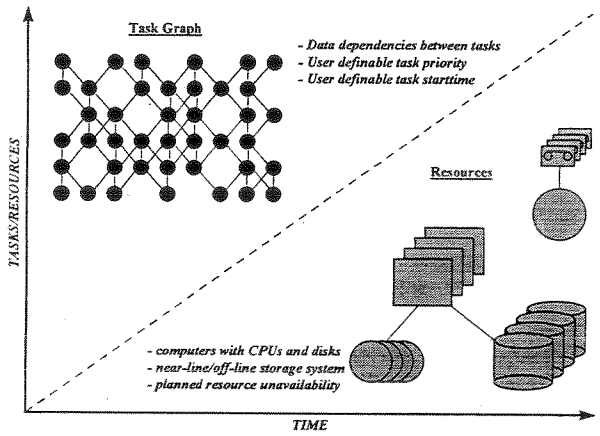


Figure 3. The overall problem domain view where tasks have to be scheduled on resources for optimal processing time and efficient resource utilization.

The challenge is the scheduling of competing tasks on resources with planned unavailability to achieve optimal processing time and efficient resource utilization. The solution must address user-definable task priority and earliest possible start time for tasks, based on predicted data arrival from an external source. These two comparison attributes must be used for task prioritization in order to support the requirements of the operations personnel. Furthermore, tasks must be scheduled around planned resource unavailability to not impact normal maintenance of hardware and software components of the system.

The approach taken to solve this problem was to implement the solution in two phases. First, research into the planning and scheduling problem domain was followed by the design and development of a prototype algorithm software application. Once the prototype was completed, the algorithm was incorporated into the Planning Workbench software component of the PDPS as the planning and scheduling engine.

Algorithm

The planning and scheduling algorithm implemented is based on the Highest Level First (HLF) List Scheduling algorithm or the Critical Path Method (CPM) [1][2]. The basic principle behind the algorithm is to assign nodes in the task graph a level (priority) which is the longest path from the node to the leaf node and scheduling the tasks to ready resources in descending priority.

As described previously, the primary use of the algorithm is to generate an initial production schedule from a strategic planning perspective and for forecasting product availability, since the actual task execution is managed by a COTS software product. Thus, a very complex and elaborate solution would not have been fully utilized.

Figure 4 depicts the overall scheme of the planning and scheduling algorithm.

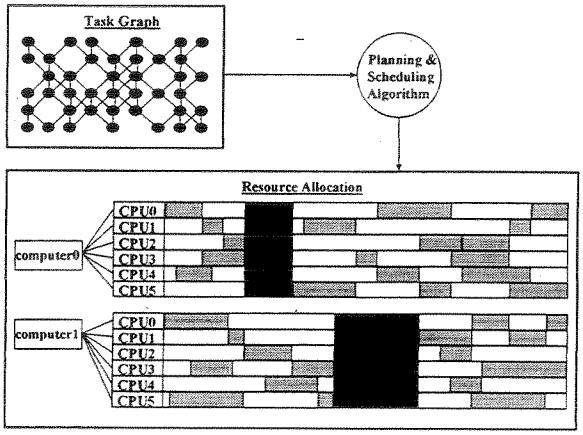


Figure 4. The algorithm sorts and prioritizes the tasks and performs non-preemptive, processor allocation of tasks around planned resource unavailability.

The algorithm performs non-preemptive, processor allocation of tasks. Communication delay between tasks is assumed to be embedded in the task size (runtime). Since the disk resource consumption is not controlled by the algorithm in real-time, no cache management of disk is designed into the algorithm. Therefore, aggregate size of all input and output data for a task is used primarily for scheduling around planned disk resource unavailability. Planned processor, computer, and dataserver resource unavailability is also taken into account during task-to-resource allocation phase in order to avoid possible resource contention between tasks and scheduled outages of resources. Finally, user-specified task priority and earliest start time for tasks based on external data arrival are used for sorting and prioritizing the task graph. Once the task graph and resources are loaded, the algorithm allocates tasks to earliest possible time slots on processors and disks to achieve optimal processing time, while attempting to load balance for efficient use of resources.

The overall data structure utilized in the algorithm is illustrated by Figure 5.

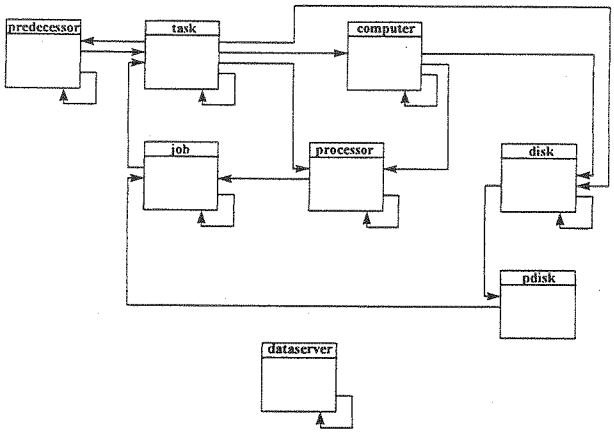


Figure 5. The algorithm uses a linked-list implementation of a weakly-connected, directed graph data structure for tasks and multi-level linked-list data structure for resources. Dataserver is a simple linked-list.

Linked-list implementation of a weakly-connected, directed graph data structure was chosen to represent the task graph for efficient traversal in loading, sorting, prioritizing, and scheduling operations. Multi-level linked-list computer hardware resource data structure was designed to logically map the available resources. Mapping of disks to physical disks was designed to support both local and network attached devices. Finally, a simple linked-list data structure was implemented for dataserver component of the resources for planned resource unavailability purpose.

There are four steps the algorithm performs in generating a schedule: load tasks, load resources and planned unavailability of them, sort and prioritize tasks, and schedule tasks on resources.

Tasks with attributes shown in Table 1 are loaded into the task graph with task as vertex and predecessor as edge.

Table 1. Task Definition

Attribute	Description
name	task name
priority	user specified priority
start time	estimated earliest start time for task based on data availability
duration	duration of task
size	aggregate size of data
predecessor	predecessor task name

Next, resources and planned unavailability are loaded into the multi-level linked-list data structure. Attributes are shown in Table 2.

Table 2. Resource Definition

Attribute	Description
name	computer name
processor	processor name
disk	disk name
size	capacity of disk
start time	resource unavailability start time
duration	duration of resource unavailability

Once tasks and resources are loaded, the algorithm sorts and prioritizes the tasks. The task graph is traversed to calculate and assign priorities and earliest possible start times to nodes. The merge-sort algorithm is used for sorting the task graph based on following comparison criteria (from highest to lowest order of precedence):

- earliest possible start time
- user-specified priority
- calculated priority
- number of immediate successor tasks

Now the fun begins. Traversing the prioritized task graph, the algorithm determines the earliest possible time slot a task can be executed on processor and disk resources. This time is verified against the dataserver unavailability to ensure no conflict exists. If it encounters a situation where the task can be allocated to more than one processor at an identical time, total processor runtime allocation is used as a tie-breaker for load balancing purposes.

The logic behind the finding of the earliest possible time slot on processors and disks is a simple one. Processor and disk resource data structures are traversed in sequence starting from time zero reference point. When a suitable time slot is located, it stops. This design ensures that resources are saturated as much as possible with running tasks so that best-fit scheduling optimization is achieved.

As tasks get allocated to available resources, linkage between task and resources are established via an abstract object called the job. This way, the generated schedule can be retrieved by traversing either the task graph or the resource data structure.

To summarize, the implementation of the planning and scheduling algorithm was presented in this section. Research into the problem domain was followed by a prototype algorithm development approach. Extensions were added to the HLF/CPM algorithm to meet the requirements of the ECS science data processing system. It was shown that (1) user-definable task priority, (2) earliest task start time based on predicted data arrival, and (3) planned resource unavailability in a distributed, multiprocessor, parallel computing environment were all addressed by the algorithm.

Implementation

The planning and scheduling algorithm described in the previous section was incorporated into the Planning Workbench software component of the PDPS. The Planning Workbench system is used to prepare production schedules in order to forecast start and completion times of data processing activities at the Distributed Active Archive Center (DAAC).

Using object-oriented methodologies, the Planning Workbench was implemented on the Hughes Delphi™ off-the-shelf C++ class library infrastructure. Delphi™ is based on a system of distributed, modular components where each component represents a distinct planning function.

The object-oriented architecture of Delphi™ is a variant of the model-view-controller (MVC) paradigm [3]. The MVC is an idealized view of the heterogeneous world of real systems. The *model* mirrors the structure of the user's conceptual model. The *view* shares the same structure and presents the model in a tangible form. The *controller* allows the user to interact with the model.

Delphi™ provides classes for an abstract model which are known collectively as the Resource Model. The Resource Model consists of resources, resource states, and all relevant resource constraints. Resources necessary for planning and scheduling are implemented as objects within the Resource Model. Besides modeling real world objects, another function of the Resource Model is to keep track of resource states over time.

In addition to resources and resource states, the Resource Model contains activities which are schedule-able entities known as tasks. During planning and scheduling, resources are assigned to activities and resource states are updated to include activities. Plans are aggregation of activities and resources that represent the science data processing production plans of the PDPS. Figure 6 describes the Resource Model in the Planning Workbench.

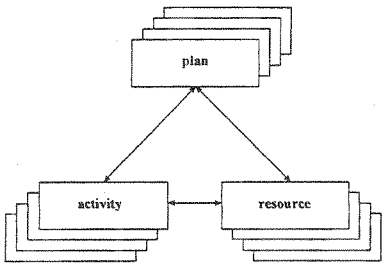


Figure 6. The Resource Model view of the Planning Workbench system where plans consist of activities and resources.

The Planning Workbench system consists of the Resource Model, the Scheduler, and the Timeline. The Resource Model constructs the resource configuration of the system by querying the PDPS database. The Scheduler retrieves user-specified science data processing tasks from the PDPS database, performs static scheduling, and directs the COTS scheduler product to execute tasks on target computers. To complete the loop, the COTS scheduler product updates the PDPS database with status of tasks. Finally, the Timeline component of the Planning Workbench system provides a GANTT style graphical view of activities and resources over time. The internal architecture of the Planning Workbench system and the external interfaces to COTS products are shown in Figure 7.

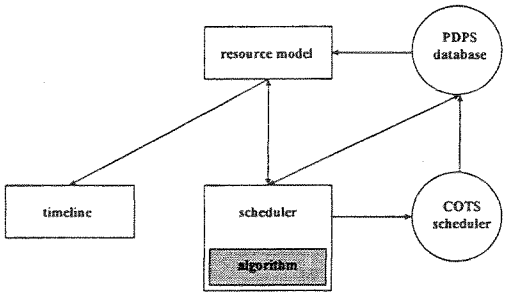


Figure 7. The inner workings of the Planning Workbench system with external interfaces to COTS products for production plan activation and management.

The implementation of the Planning Workbench system involved the incorporation of the planning and scheduling algorithm as the engine and the development of concrete classes which represent objects in the PDPS domain such as computers, processors, resource states, resource unavailability, and science data processing requests.

Currently, the PDPS is deployed and operational as a Science Software Integration and Test (SSIT) testbed at four Distributed Active Archive Centers (DAACs): the EROS Data Center (EDC), the Goddard Space Flight Center (GSFC), the Langley Research Center (LaRC), and the National Snow and Ice Data Center (NSIDC). It is the basis for the enhanced ECS which is scheduled for deployment in 1998.

Conclusions and Future Work

This paper has presented the implementation of the planning and scheduling algorithm to satisfy the science data processing requirements of the ECS. The overall problem domain was described, and details of the algorithm development were presented. A heuristic approach was taken to design the algorithm, and systematic software engineering steps were followed to implement the final solution.

The algorithm was designed for planning and forecasting purposes. To take full advantage of the effectiveness of the algorithm, and to realize its potential as a production engine, it should be enhanced to perform dynamic scheduling where the system manages task execution in order to perform decision making in real-time.

Acknowledgments

This work was funded by NASA Contract NAS5-60000. The authors would like to express appreciation to Tom Atwater for comments.

References

[1] El-Rewini, H., Lewis, T., Ali, H. 1994. *Task Scheduling in Parallel and Distributed Systems*. Prentice Hall, pp. 56-81

[2] Pinedo, M. 1995. *Scheduling Theory, Algorithms, and Systems*. Prentice Hall, pp. 61-92

[3] Goldberg, A. July 1990. Information Models, Views, and Controllers. Dr. Dobb's Journal, pp. 54-60

Author Index