N90-22298

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

by

William C. Eggemeyer
Jennifer W. Cruz

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

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

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

INTRODUCTION

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

PLAN-IT

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

Plan-IT Background

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

SpaceLab

SpaceLab was a multi-year effort requiring extensive code additions to adapt Plan-IT to operate within an already existing scheduling system, called Experiment Scheduling Program (ESP). Plan-IT's task for this application was to permit the user to tweak an already existing schedule either graphically by manual tweaks or by algorithmic strategies specially coded for the SpaceLab problem domain. The SpaceLab experience demonstrated that the Plan-IT approach could handle their scheduling problem with a few caveats. The

drawbacks were that adapting Plan-IT to SpaceLab's particular problem domain took an excessive amount of time and required extensive coding modifications to Plan-IT's internal representations. These drawbacks indicated that Plan-IT's internal structures were not robust enough for easy adaptation to different problem domains.

Space Station Power Scheduling Proof of Concept

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

Deep Space Network (DSN) Application

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

Comet Rendezvous Asteroid Flyby (CRAF) Demonstration

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

Telescience Demonstration

This demonstration was a joint JPL and Goddard Space Flight Center (GSFC) effort. GSFC provided the user-interface into the scheduling system, and JPL provided PLAN-IT, the scheduling tool. This task demonstrated that additional research into the field of peer teleconferencing is greatly needed. Allowing multiple users to concurrently interact from their home institutions on a

schedule of activities raises a few issues: 1) how are multiple user simultaneous and cooperative edits to the schedule to be handled; 2) how is control of the scheduling session handled; 3) how should security be handled to guarantee privacy among the users, and; 4) how should the database be maintained. These and other Telescience issues will be addressed in a proposed future effort.

Knowledge Gained from Past Experiences

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

Visualization and Adaptation Issues

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

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

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

Constraints and Representation Issues

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

In many of the problem domains there were requirements for handling temporally complex activities. Plan-IT's activity representation by only frames and slots [1] was not flexible enough for adequately representing these complex types of activities. An example of a complex activity is a generic request or a cyclic where an activity is supposed to occur multiple times every

so often in time. Tweaking components of such a structure may have repercussions on the other components depending on the temporal dependencies involved. Defining complex activities in Plan-IT was a complicated and tedious programming task, leading to the need for a simpler and more robust method.

Scheduling Issues

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

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

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

PLAN-IT-2

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

Processes in Plan-IT-2

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

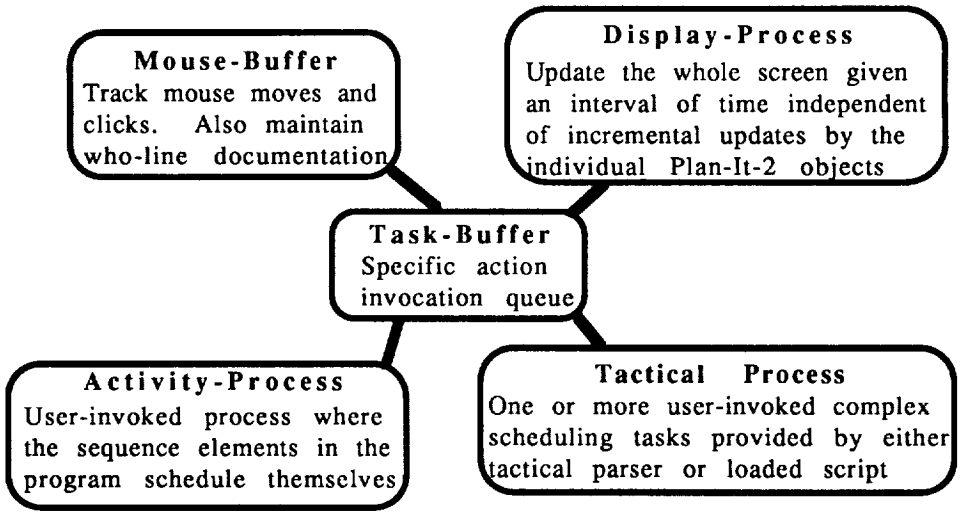


Figure. 1 Plan-IT-2's Processes

Process Interaction

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

Scheduling Processes

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

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

conflict. So, instead of moving, spawning and slinking on each activity instance before going to the next instance in the set, Plan-IT-2 applies each action to the whole set of activity instances before going to the next action. This demonstrates the relative ease for a user to generate scheduling algorithms at his own level of understanding, rather than by unintelligible predefined hard-coded routines.

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

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

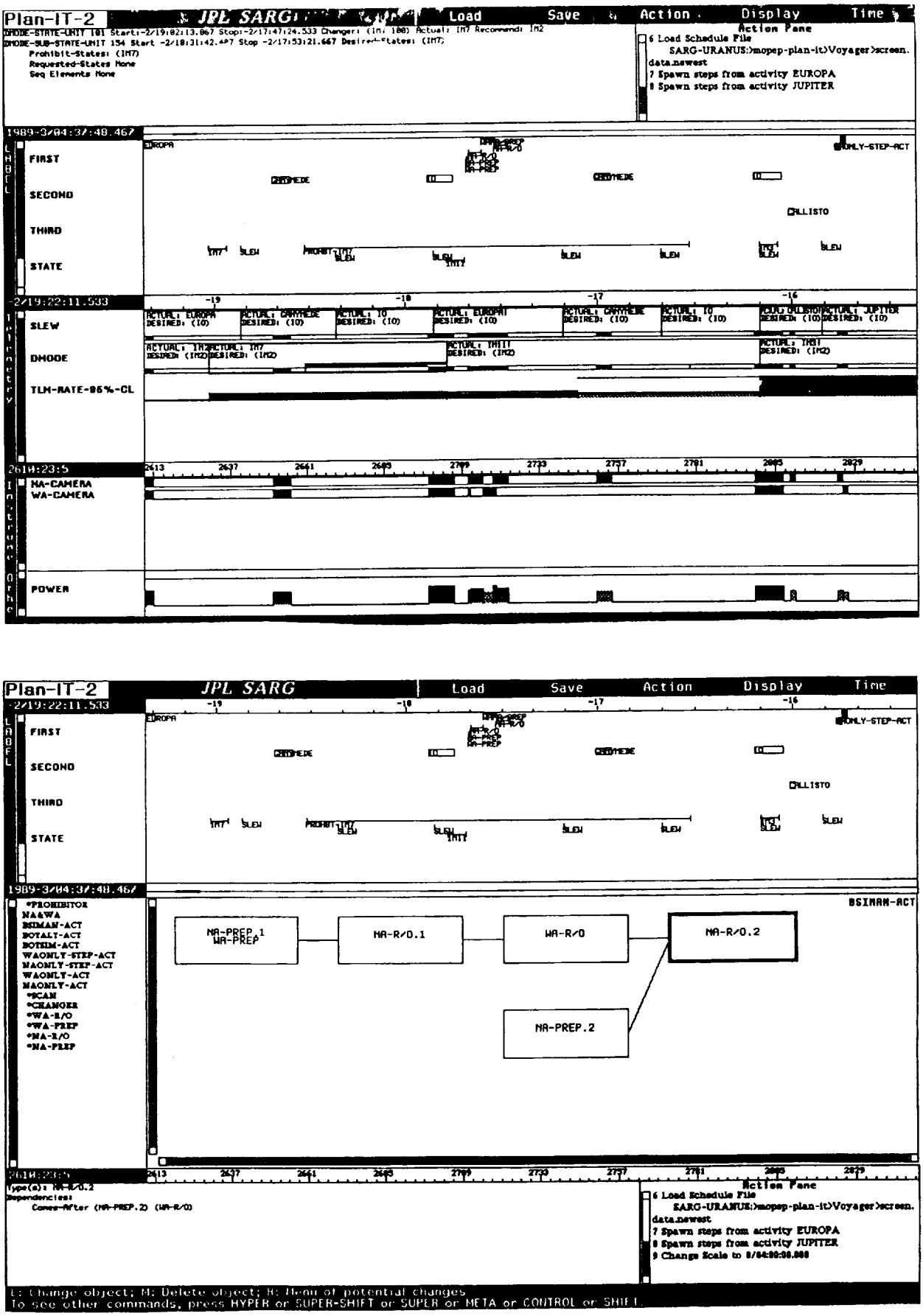
File Types

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

Other Interaction Capabilities

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

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



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

Resource Constraint Capabilities

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

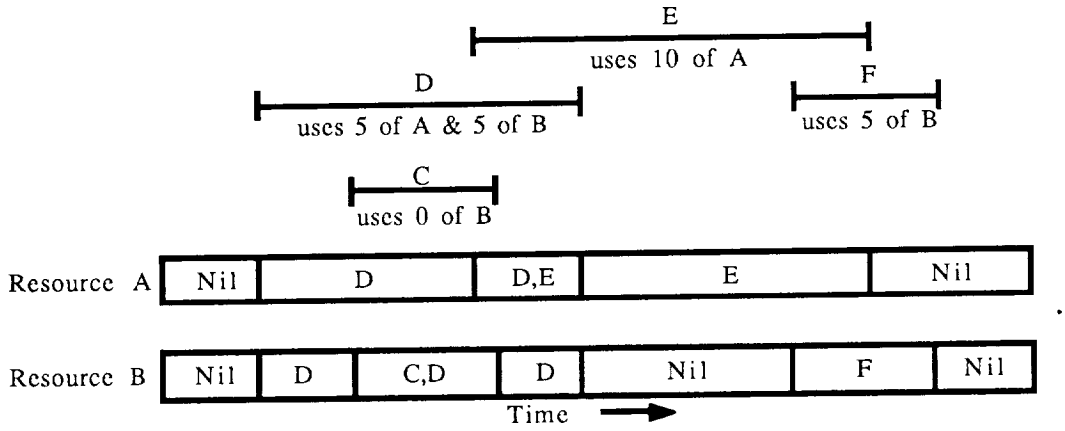


Figure. 2 Activities C, D, E and F monitored by resource constraints A and B

Constraint Dependency Mechanism

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

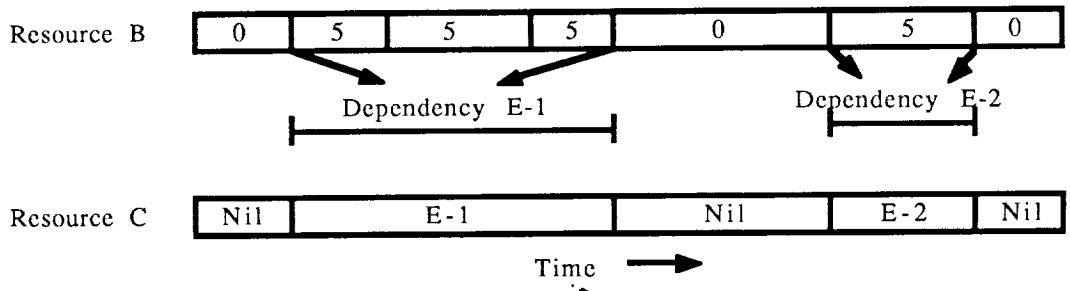


Figure. 3 Uni-directional generated dependency events monitored from B to C

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

Concurrencies

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

Non-Depletables

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

Depletables

Depletables are a consumable type of resource constraint that DOES NOT restore itself when not in use. As with non-depletable types there are several forms of depletables. The simplest called *depletable* starts with a fixed amount of

something and through a step-wise fixed consumption the amount gradually becomes depleted. A simple example of this is modelling fuel usage on a spacecraft in a step-wise fashion. The spacecraft starts out with a fixed amount and as it is being consumed by the thrusters during maneuvers it gradually becomes depleted.

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

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

States

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

Activity Representation

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

Time Slot

Time representation has been separated from the rest of the slots to give the user a greater amount of flexibility for influencing the activities' choice of actions. The time slots in Plan-IT-2 permit the user to define temporal flexibility of the activity instances in the schedule. The time slots vary from

just having a start, stop and duration value to having a flexible duration with multiple time windows containing multiple preference choices. Another new Plan-IT-2 feature is the ability to tag the time system type to the data during input so that it can be saved in the same time format.

Generic Slots

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

Table 1. Thirteen different Plan-IT-2 Slots

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

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

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

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

Example 1. Some Slots for the Voyager Problem Domain

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

Activity Types

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

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

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

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

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

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

Activity Node Network Structure

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

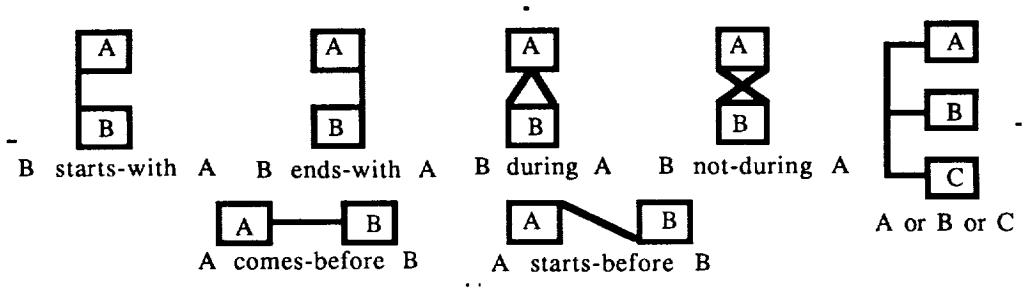


Figure. 4 The seven nodal relationships for an activity network

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

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

Example of Defining Activity Types

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

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

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

Example 2. Step Definitions for an Imaging Activity

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

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

Example 3. Activity Definition for Intermediate Level of Detail

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

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

Example 4. Most Abstract Activity Definition

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

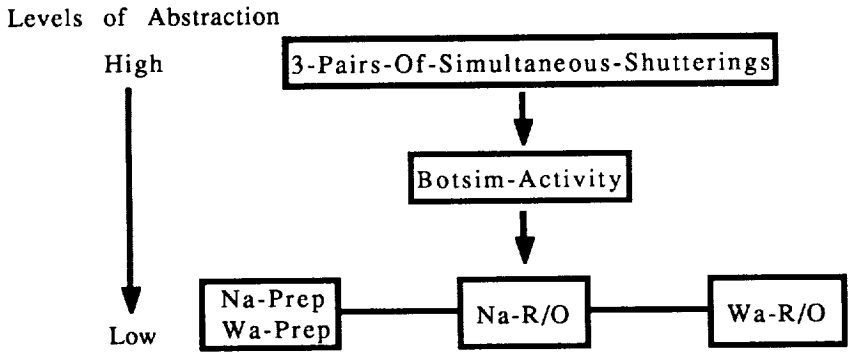


Figure 5. Abstract layering of 3-Pairs-Of-Simultaneous-Shutterings

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

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

Generic Action Definition

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

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

User modifications on the activity structures are handled in much the same way. For example, if a user moves an intermediate-level abstract activity instance by the mouse. If this instance, being a child to a complicated activity

structure, was moved beyond its allowable flexibility definition within the node network, or moved into another state that caused it to change, the whole layered structure of the node networks updates itself appropriately.

Remaining Plan-IT-2 Objects Used in Scheduling

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

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

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

Plan-IT-2 Mode of Execution

Plan-IT-2 uses conflicts in the schedule only as motivators for taking scheduling actions. The conflicts themselves are no longer used to determine the success of a scheduling action. The monitoring of success of an action is left to both the activities themselves and the user monitoring the program. The execution of the scheduling actions relies on viewing across multiple constraint timelines based upon the opportunistic view presented by the constraints, merged together by the scouts for the activities. Unlike the original Plan-IT, there is no form of global measurement of goodness for the

schedule by the constraints. Presently, Plan-IT-2 executes these actions serially from the task buffer. When Plan-IT-2 is completed an attempt to parallelize the automatic determination and execution of non-interfering actions will be made.

SUMMARY

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

ACKNOWLEDGEMENTS

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

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

REFERENCES

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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