JIT Planning: an Approach to Autonomous Scheduling for Space Missions[1]

Pierre F. Maldague, Adans Y. Ko
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109-8099
818-354-0152
maldague@jpl.nasa.gov, ako@jpl.nasa.gov

Abstract— The purpose of this paper is to report the results of a one-year project aimed at demonstrating the concept of "Just-In-Time" (JIT) Planning. The project is based on existing JPL software: Apgen (a resource-based activity planner), Spice (a library for accessing and processing trajectory information), and DARTS Shell (a S/C attitude control and simulation package). The distinguishing feature of our approach is that planning is done in real time, concurrently with execution. In this way, the planner has access to up-to-date information and is able to accommodate unforeseen variations in S/C or external conditions. By inserting small amounts of "software glue" between these three components, we achieved a smooth transition from deterministic execution of a time-ordered command sequence to an adaptive system that responds in closed-loop fashion to events predicted by the simulator. The main benefit of this approach is to provide an adjustable level of autonomy, ranging from conventional commanding to full autonomy. A significant by-product of the JIT approach is that it can eliminate the need for detailed S/C models, which are replaced by actual or simulated real-time data.

TABLE OF CONTENTS

1. INTRODUCTION

The purpose of this document is to report the results of a one year project that was funded as a Continuous Improvement Proposal by both the Telemetry and Mission Operations Technology (TMOT) and the Center for Space Mission Architecture and Design (CSMAD) at JPL. The central concept behind the proposal was "Just-In-Time" (JIT) Planning, an idea which has been proposed by other authors in a somewhat different context [1]. The nuts and bolts of the proposal consisted in combining existing tools for planning, sequencing, navigation, and S/C attitude control modeling with the minimum amount of glue necessary for these components to work in harmony.

The conceptual framework for the work described here owes much to an ongoing project at JPL called X2000. This project is aimed at showcasing a number of new technologies in the context of near-future space missions. In particular, X2000 has been interested in approaches to S/C commanding that allow more autonomy than in the past. Many of the ideas described in this paper arose during discussions of the X2000 requirements.

The primary focus of our discussions was to discover how one could evolve the existing Planning and Sequencing (P and S) system currently in use at JPL into one that is more in tune with autonomy requirements. In looking for an answer, we did not spend a lot of time on the interesting but somewhat theoretical issues of data structures and control strategies for autonomous systems. We paid more attention to the practical issues that one runs into when attempting to design a ground data system that can support a partially autonomous S/C. We also wanted to demonstrate whatever concepts we came up with within a short time, and this forced us to stay fairly close to the tools that we are already using now.

Within these limitations, our work shows that a continuous transition between the existing, traditional commanding system and a more autonomous system is possible. Such an approach will demand new skills of ground data systems personnel, especially in the area of "gluing" together planning and modeling engines in a real-time environment.

This paper is organized as follows. In Section 2, we attempt to look at the interacting ground-flight system as a single entity, and we show how the current architecture of the ground-flight system can be evolved into a system that reduces duplication and enables closed-loop commanding. In Section 3, we present our JIT implementation in the context of an observatory-like mission such as SIRTF. Section 4 presents our conclusions as well as possible future extensions of the present work.

2. THE INTERACTING FLIGHT-GROUND SYSTEM

The purpose of this Section is to paint the "grand dream" that motivated us to undertake the JIT planning work. The ideas presented here are somewhat speculative, and they represent "thinking in progress" rather than the finished,

[1] 0-1234-5678-0/99/$5.00 © 1999 IEEE

optimized product of a well-organized engineering effort. We present these ideas in their unfinished state because we think that mature designs are not yet available, yet there is a need to show a "big picture".

As indicated in the introduction, we differ from other approaches because we believe that in the short- to medium-term, achieving autonomy will be a systems integration problem, not a software technology problem. To support our view, we will present in this Section a simplified picture of where we are today in terms of spacecraft commanding strategy, and how we can migrate towards the future through incremental steps.

Traditional Sequencing: Time-Tagged Commands

The interaction between the Flight and Ground sides of a space mission is governed by two documents: on the uplink side, the "Command Dictionary" (CD) contains a list of all the commands which the S/C understands, along with their parameters and a description of what they do. On the downlink side, the Telemetry Dictionary (TD) specifies the nature and format of the data that will be transmitted by the S/C to Earth for analysis. Since we are concerned with commanding, we concentrate on the uplink side.

The primary task of the uplink team in a traditional mission is to prepare sequences of time-tagged commands for uplink to the S/C. Schematically, such as sequence can be represented by the diagram shown below.

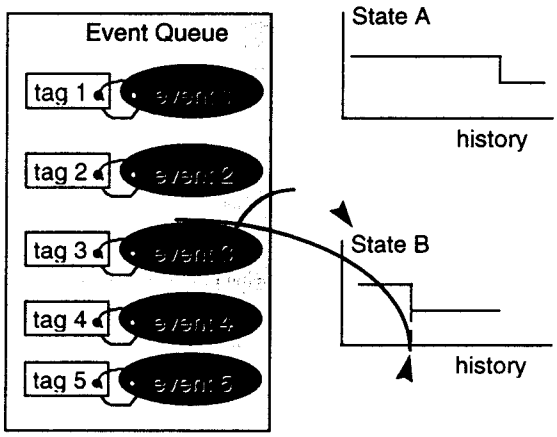

Fig. 1: Traditional sequence of time-tagged commands

A sequence is a special case of a more general "event queue", which could contain (as is usually the case in ground simulations) events beyond the mission team's control such as planetary occultation or downlink opportunities involving specific Deep Space Network ground stations, as well as executable commands. The flight version of the sequence usually contains commands only; these commands are a binary representation of the human-readable form used on the ground.

In any case, the purpose of the sequence is to modify the state of the S/C in a desirable way. As illustrated in Fig. 1, each command can have an influence on the various state variables that describe the complete state of the S/C. In ground simulations, changes in state variables are simulated through "modeling" of the Spacecraft's behavior. The history of each state variable is maintained by the simulation. These simulated histories are then scanned for possible violation of flight and mission rules.

Sequences are often represented on a timeline, which makes the unfolding of events and activities more intuitively obvious. Fig. 2 below illustrates this in the context of power usage and shows how a timeline might represent activities that use power as well as the state of the power resource itself and of related resources such as battery charge.

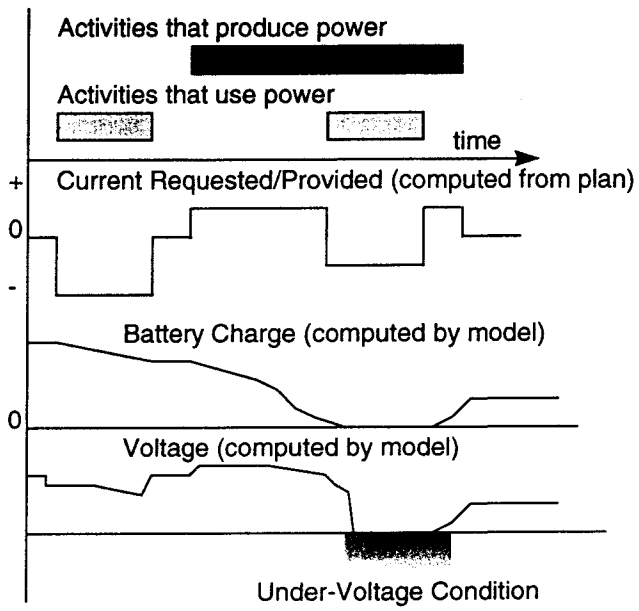

Fig. 2: Activity and resource timeline

On-board the S/C, commands are stored in a special area of memory reserved for the sequence. When the time comes to execute a given sequence (several sequences can execute in parallel), the Command and Data Handler subsystem clocks out the commands at their prescribed time and dispatches them to the flight software for execution.

Differences between Flight and Ground

It is clear from the above discussion that although they share a common origin, sequences are handled differently by the flight and ground systems. The following Figure highlights the differences:

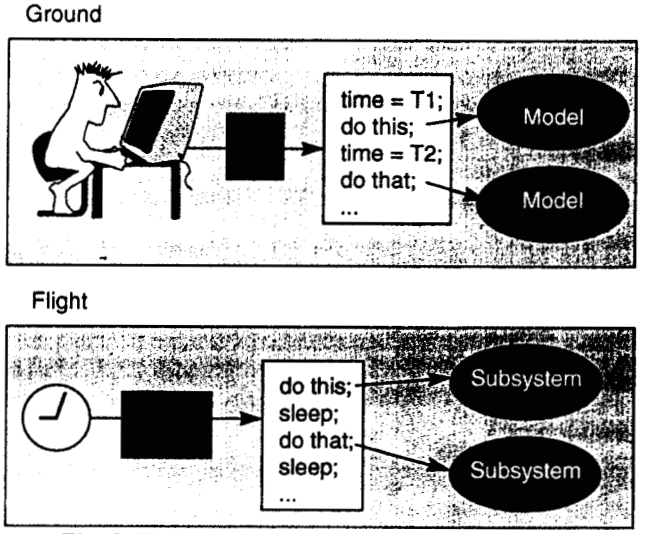

Fig. 3: Flight and Ground Systems handle command sequences (yellow rectangles) in different ways

On the ground, the simulation of S/C events is carried out by "setting" the simulation time equal to the tag of the next command in the sequence, at which point the command is executed and the time is set to the next tag in the queue. There is usually no reason why the simulation should wait in between time tags; the simulation is usually carried out as fast as the processor in use will allow, so as to provide the human analyst in charge of the simulation with the fastest possible response.

On-board the S/C, the situation is different because commands should only be executed at the time specified in their time tags. In between commands, the command processor is basically idle, although of course considerable activity may be taking place in the various subsystems while waiting for the next command.

One of the biggest challenges faced by P and S personnel supporting a space mission is to come up with models that accurately represent the behavior of the S/C. Unless these models are accurate, one cannot guarantee that a proposed sequence will not cause any harm to the S/C or to the mission. The ultimate tool in sequence verification is the S/C testbed, which reproduces in hardware the precise configuration used on-board. Unfortunately, S/C testbeds are not only costly, they are also impractical in all but the most critical circumstances because they typically run at the same speed as the S/C. A good planning and sequencing tool must be able to provide a functional simulation of S/C behavior that runs of the order of 1,000 times real-time in order to provide the fast turnaround time required for efficient operations.

How can we bring Flight and Ground closer together? Fig. 4 below shows how some commonality can be achieved between the two simply by recognizing that both rely on a time clock. Because the ground software does not require 100% fidelity, it can run significantly faster than its flight counterpart, as suggested by the different clock rates in the figure. The figure basically suggests that the real-time C&DH (Command and Data Handling) subsystem could be made common between flight and ground. This is not a trivial assumption, because the flight software typically runs under a real-time OS while the ground system runs under an OS more commonly found on engineering workstations or personal computers. Some success has been achieved in this area, and several JPL applications have been adapted to run both under the VxWorks real-time OS (Wind River Systems, Inc.) and in workstation environments such as Solaris (SUN Microsystems Inc.).

The next step in the path to ground-flight integration is to address the adaptation issue. So far, we haven't said much

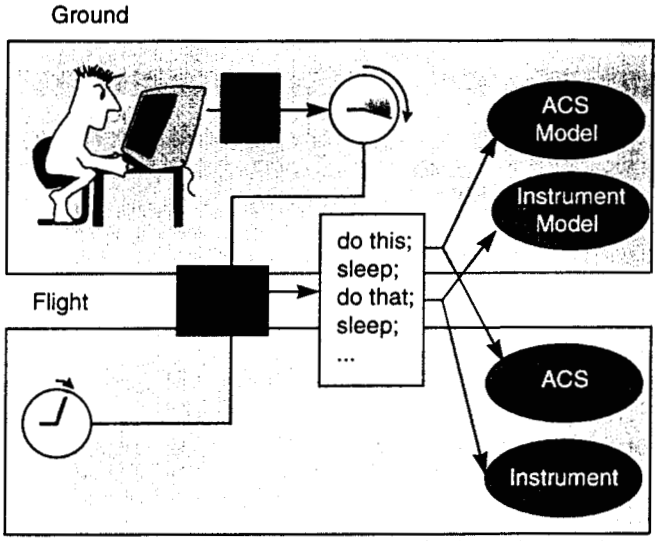

Fig. 4: Better integration between flight and ground

about the objects that commands act on: subsystems on the flight side, S/C models on the ground side. Traditionally, each new space mission has had its own brand of flight software; there was not much commonality between successive missions, and re-inventing the flight software for each mission was not a major issue. Two factors are driving flight software in the direction of more re-use from one mission to the next: first, radiation-hardened flight computers available today have grown tremendously in sophistication and power, inviting S/C engineers to make their flight s/w ever more complex. Second, instead of a few major missions, JPL is increasingly looking at many smaller missions. Such missions have much smaller budgets than

their predecessors, which makes it mandatory to re-use as much s/w as possible.

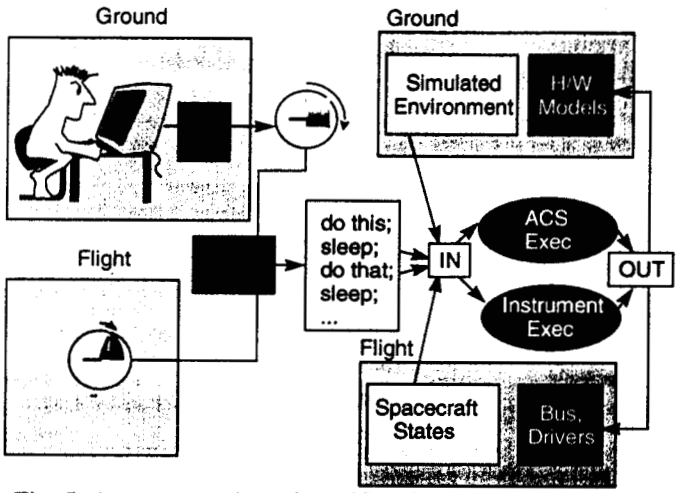

Fig. 5: A representation of a unified flight-ground system in which the front-end (API) of each subsystem has been isolated and made common with the ground. The API talks to models or simulation S/W on the ground, and to the flight S/W on board.

This trend towards multi-mission, reusable software was started a number of years ago in the ground data systems area. As a result, the main tools used for ground sequencing and commanding at JPL are now fully adaptable, in the sense that the software itself does not change from one mission to the next. What does change from one mission to the next is the set of "adaptation data" that the software needs in order to represent activities and commands that relate to a specific S/C.

On the flight side, although the idea of re-using software is not new, the concept of multi-mission software has not reached the same level of maturity as on the ground. With the advent of near-compatible ground and flight Operating Systems, it should be possible to achieve the same level of multi-mission ability on the flight side as on the ground. This is illustrated in Fig. 5 above. The new element here is that both flight and ground systems are commanded through the same "API" (Application Program Interface). By API, we mean a software interface which can be called directly by the commanding subsystem and which delegates its tasks to actual subsystems (on board), functional simulations (ground) or S/C models (ground).

So far in our discussion, we have not said anything about commanding strategy. In fact, our diagrams were pretty much consistent with the traditional sequencing and commanding methodology of previous space missions. To support autonomy, however, we need to take into account the fact that the S/C has on-board planning capabilities. This fact is emphasized in Fig. 6 below.

The new element in this picture is the planner, shown as the box at the top center of the figure. This is in reality the on-board planner; there may be other programs in use on the

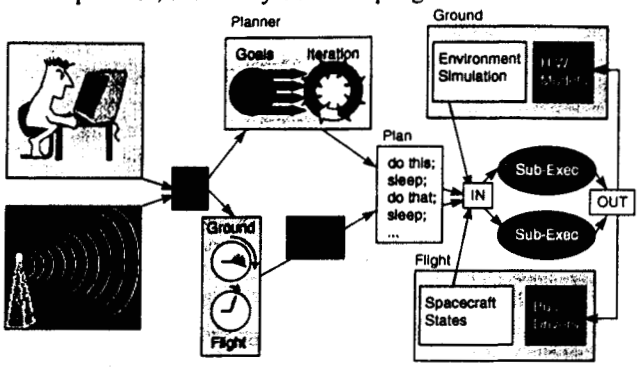

Fig. 6: Unified flight-ground system featuring an on-board planner. The building blocks of both systems have been made common, except of course for the hardware-level interfaces which need to be simulated on the ground.

ground that have planning capabilities. In fact, it is likely that in the first implementations of this architecture the on-board planner won't have much in common with the planning systems found in the AI literature, but will instead resemble a sequence with embedded logic in it, along the lines of Lockheed-Martin's Virtual Machine Language (VML).

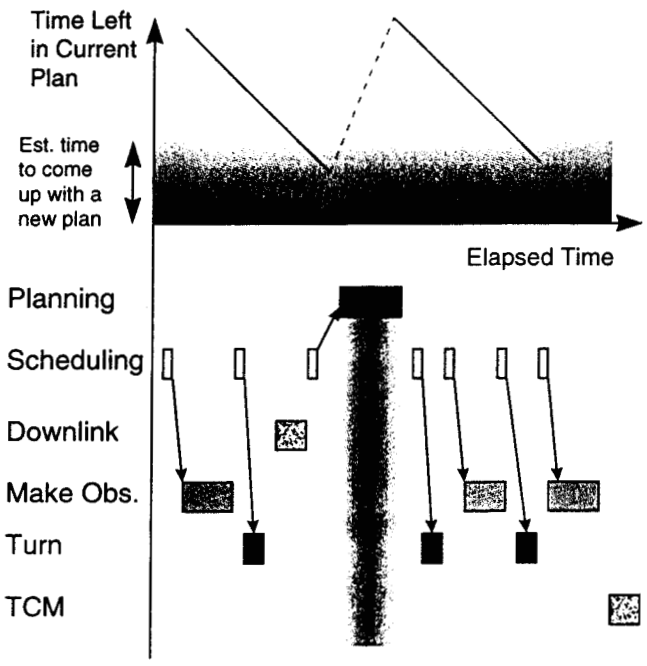

Fig. 7: This activity timeline shows that time needs to be allocated for on-board planning. The key resource is "time left in current plan", displayed at the top. When that resource drops into the danger zone, a "Planning" activity needs to be scheduled (green rectangle).

To conclude this section, we return to the timeline illustration of a S/C activity plan. The operation of an integrated ground-flight system such as illustrated in Fig. 6

needs to take into account the fact that the planning activity itself can take significant amounts of time. Therefore, it needs to be included among the S/C variables that are continuously monitored by the flight S/W, as shown in Fig. 7 above. Note that both planning and scheduling activities are accounted for in this timeline. We are implicitly assuming that scheduling can be done on a short-term basis, as would be the case for an observatory that is given a list of desired (unconstrained) observations. It is anticipated that this type of short-term scheduling does not take much time (see the orange rectangles in the figure). Planning will generally take longer, first because plans extend over a longer period of time, second because planning may involve more iteration if an attempt is made to optimize the on-board schedule.

3. Present Implementation of JIT Planning

In implementing our approach to JIT planning, we first decided to limit the scope of the problem so as to give ourselves a chance of producing something tangible within a year. We decided to limit ourselves to tools that already exist and are readily available within the JPL community: DARTS Shell, a S/C Attitude Control Subsystem (ACS) simulation tool [2], Spice, a library of navigation aids [3], and Apgen [4], a resource-based activity planner for space missions, , and. Apgen was chosen because it can be easily adapted through external text files, and because it can support interaction with other processes through UNIX sockets in real time. Although Apgen is a planning tool in use for current space missions, we used it as a short-term scheduler, not as a long-term planner. The main reason is that when used in automatic planning mode, Apgen relies on the adapter for avoiding scheduling conflicts, not on built-in conflict avoidance algorithms as some more advanced planners do. Developing sturdy scheduling algorithms is possible but time-consuming. Since we wanted to concentrate on systems integration, we decided to keep the algorithm development effort to a minimum.

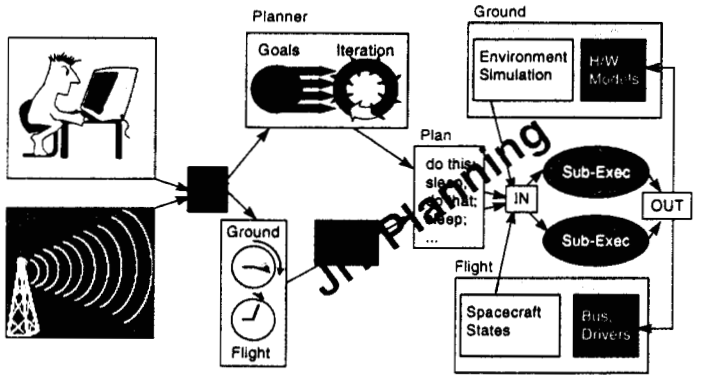

Fig. 8: JIT planning addresses only a subset of unified flight-ground integration issues: real-time operation, commanding system, S/C modeling, and common API. Other issues such as planning optimization and hardware sensing are not covered.

Alternatively, we could have used a planner that has more built-in capabilities for automatic production of conflict-free schedules, such as the Remote Agent planner developed at ARC and JPL. However at the time we started our JIT effort we did not have sufficient expertise in the use of this tool. We therefore decided to restrict ourselves to short-term scheduling only. We will re-visit the important issue of planning technology in the next Section.

Fig. 8 above illustrates how our implementation relates to the general picture painted in the previous Section. We were particularly interested in providing answers to the following questions:

1. What does it take to operate a sequence planning tool in a real-time mode?
2. What adaptation skills are required to write real-time applications?
3. What technology should be used to provide the "glue" between the applications that contribute to this system?

Planning Context: Observatory Scheduling

Although our JIT work was clearly experimental in nature, we wanted to have at least some vague relevance to an actual space mission. We based our adaptation on an early study of the SIRTF (Space Infrared Telescope Facility) mission that took place a year ago. This study focused on a strategy in which the telescope was given a list of a few hundred observations to be carried out over a period of about one month. The observations had been designed so that no flight rules (such as pointing away from the Sun) would be violated as long as an observation was carried out within the indicated 1-month period. Just about the only constraint that had to be verified was that sufficient downlink time was allocated to avoid overfilling the solid-state recorder (SSR) on-board the telescope.

The original implementation of the SIRTF prototype was quite typical of the situation depicted in Fig. 3, in that the activity plan was entirely based on ground-based models of S/C behavior. First, a list of downlink opportunities were provided to the planner. These opportunities are decided ahead of time and have to do with scheduling priorities between the Deep Space Network and any active space missions. Next, a list of pre-processed requests for observations are loaded. As indicated above, the purpose of the pre-processing was to ensure that no conflicts would be generated regardless of which observations were picked by the planning algorithm. Next, a simple ACS model was activated. All this model does is keep track of the current attitude of the S/C (expressed as right ascension and declination) and consult an external module (Seq_pointer at the time) to find out how long it would take to slew to the RA/DEC of a proposed observation, or to Earth for a

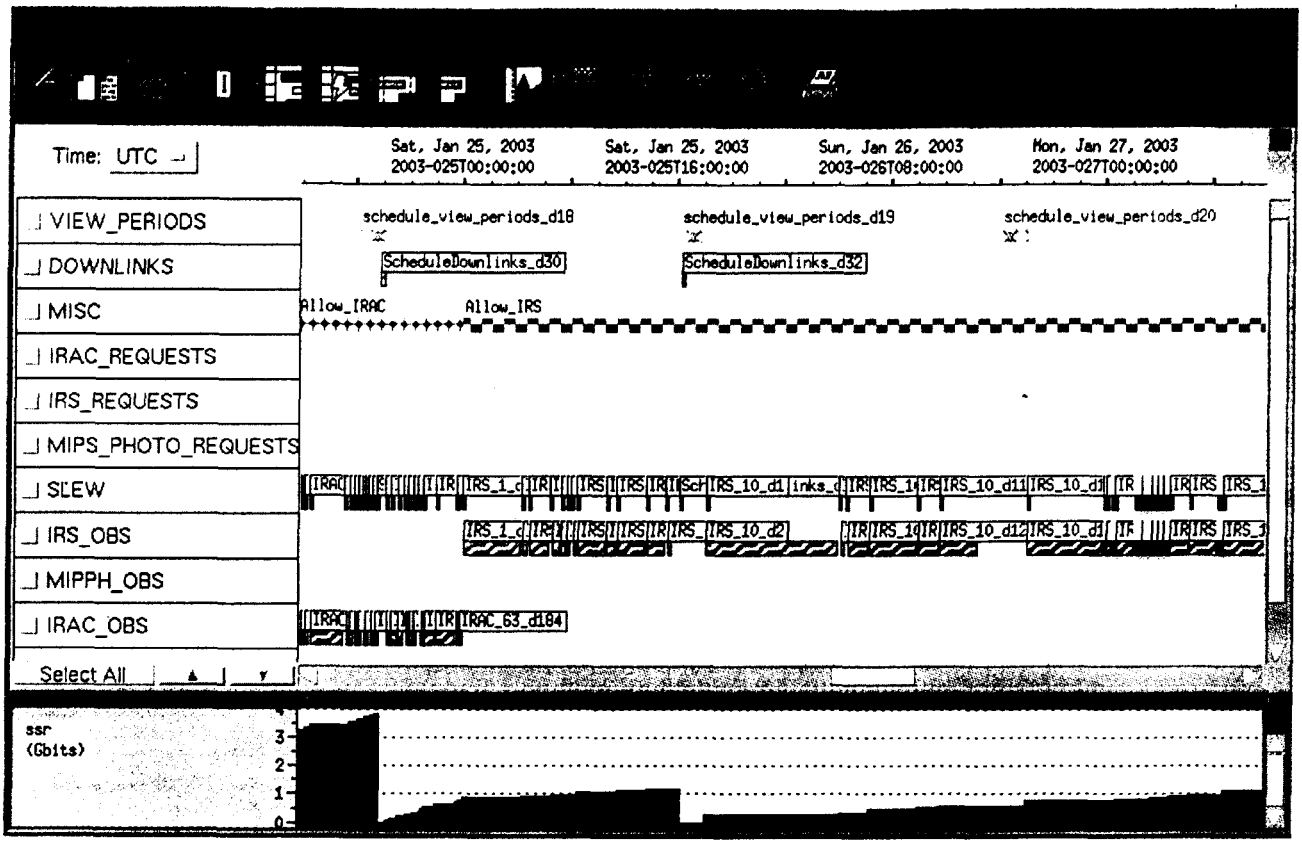

Fig. 9: output of JIT Planner for the SIRTF Prototype Scheduling Problem. The SSR fills up as MIPS and IRAC observations are scheduled, and is emptied during downlink activities. Each Slew requires a turn activity as shown on the ACS line.

downlink activity. The scheduling algorithm kept track of the amount of data stored in the SSR and knew about the data rates that could be sustained during downlink. Since each observation request contained a precise duration for the proposed observation, the scheduling algorithm was able to determine whether there was time to perform a given observation and still guarantee enough time during the downlink opportunity for emptying the SSR.

Without getting into too much detail, let us summarize the main building blocks of the observatory scheduling algorithm used in the prototype. This will help set the stage for the real-time, JIT approach discussed in the next paragraphs. The planning strategy relies on the following items:

1. *downlink opportunities are represented by* fixed-time activities occurring at 12-hour intervals. In our prototype, the planner did not have to make use of the downlink opportunity, and if it did decide to make use of it, it did not have to use up the entire time allocated
2. instrument scheduling is done using a very simple "rule of thumb": exclusive use of the spacecraft is granted to one of three major instruments for three days. Which instrument should be in charge is determined by a simple rotation algorithm, so that each instrument gets exactly one third of the total observation time available. To enforce this strategy, three mutually exclusive enabling activities called "allow_IRAC", "allow_MIPS", "allow_IRS" were introduced. An instrument can only be used while the corresponding enabling activity is active.
3. the planner is given a list of observations for each one of the three major instruments. Each item in the list specified the Right Ascension and Declination of the target as well as the duration of the desired observation. Observations were arranged so that no geometric constraints would be violated as long as an observation was scheduled during a given 30-day period.
4. the planner has complete freedom to schedule observations from the list in any way it wants, as long as it does not overfill the SSR.

Needless to say, this is a much simplified representation of reality. No attention was paid to the problem of minimizing overall slew time (the "traveling salesman" problem), nor to the problem of scheduling engineering maintenance activities such as dumping angular momentum from the reaction wheels etc. In spite of this, we found it helpful to concentrate on something that has at least some vague resemblance to a real problem.

Paving the Way for Real-Time Scheduling

Converting the SIRTF prototype to the JIT approach took a number of steps. Our high-level goal was to eliminate the slew-time model used in the prototype and replace it by DARTS Shell, which actually simulates the operation of he ACS. In order to use this simulation program, however, we needed a number of things:

- the ability to specify the S/C attitude as a solid-body rotation matrix (a quaternion, actually)
- the ability to run Apgen in real-time mode
- establish connections between Apgen and external processes
- a modified adaptation that would replace the old slew models by calls to the DARTS Shell

To achieve the first objective, we linked Apgen to a general-purpose navigation library called Spice. This link gave us access to not only the ability to translate target positions in (RA, DEC) into S/C attitude expressed as quaternions, but also to some very basic pieces of information such as the trajectory of the SIRTF, planetary ephemerides ant many other useful data.

The second item, running Apgen in real-time mode, can be explained as follows. Much of the work done by Apgen is to evaluate the impact of a given activity plan on the resources that have been defined by the adapter. The paradigm used by Apgen is the discrete event simulation model, which had been implemented successfully in Apgen's ideological ancestor, Seqgen. Seqgen and Apgen were developed in response to the need for sequencing tools that would support multiple space missions and are now used by several space missions at JPL as well as other institutions.

According to the discrete event simulation model, the impact of a S/C activity on the resources is evaluated through a series of "events" that are generated whenever the activity consumes or otherwise affects the resource. Each event has a precise time tag associated with it. In Seqgen and Apgen, this time tag is evaluated by "expanding" the activity into its constituent commands. A consumption event can be looked at as a program fragment that has a start time associated with it. Note that the program fragments are not executed until after the entire set of activities has been expanded into commands. This is because the effect of commands on resources needs to be evaluated in time order; it would make no sense to turn an inactive switch OFF at time t2, then turn the switch ON at a previous time t1 less than t2. Therefore, Seqgen (and likewise Apgen) stores all consumption events into an "Event Queue", then sorts the queue in time order, then evaluates the effect of the events on the resources.

It is easy to see how the simulation strategy just described can be modified to interact with real-time processes. Since events have already been arranged in a time-ordered sequence, the simplest way to introduce real time into the simulation is to synchronize the execution of these events with an external clock. From a programming perspective, this can be done simply by inserting a "wait" instruction in the loop that scans the event queue; the argument of the wait instruction is the time difference between the time tag of the next event in the queue and the current time.
Note that this time difference could in principle become negative. This would be the case in a simulation that is so complex that the CPU would not have time to complete the simulation of an event by the time the next event in the queue needs to be processed. We don't know of a generally way to express "model complexity", but the following table summarizes our experience with planning and simulation software. The data in the table reflects our experience with state-of-the-art hardware environments such as SUN Ultra-2 engineering workstations as well as PC's running 300 MHz CPU's.

Type of Model	Max. Speed (multiple of real time)
High-level functional model	> 3600x
Detailed functional simulation	7x
Full-scale, bit-level simulation	1-2x

Opening up the modeling loop to synchronize it with external events allows us to tie the planning process to real-time data such as the state of the spacecraft at any one time. In our case, we concentrated on the state of the ACS subsystem as modeled by the DARTS Shell system. However, in order to carry out scheduling we needed to endow Apgen with one more capability, namely the ability to expand an activity concurrently with the modeling process. The following example will clarify this requirement. Suppose that the decision has been made to schedule an observation. This means that we must expand the requesting activity R into its constituents, which typically include the following:

- a pre-conditioning activity P1
- a turn T
- a post-conditioning activity P2
- an data collection activity D1 that instructs a specific instrument to acquire the observation data
- a data transfer activity D2 that transfers the data to the SSR

In traditional sequencing, R would be expanded into all its constituents based on fixed expansion rules that reflect the ground personnel's understanding of how the S/C operates. In particular, the duration of the turn activity T would be computed based on formulas supplied by the S/C ACS engineer. In the JIT approach, such formulas are replaced by direct interaction with the ACS simulation tool, DARTS Shell in our case. Since we don't know when the turn will be complete, we must put the expansion of activities that follow T on hold until T completes. This represents a change in how the expansion software operates within the planner; we refer to it as "concurrent expansion" in the discussion that follows.

The Apgen implementation of concurrent expansion is actually a two-phase process. The first phase takes place

before the real-time simulation is started. During this first phase, background activities that are "cast in stone" are expanded in the traditional way. This includes things such as downlink opportunities and engineering maintenance activities. The reason for processing such activities first is that they provide the necessary context in which scheduling decisions can be made. For instance, in order to decide whether or not to start an observation, the S/C needs to know how much time is available before the next downlink opportunity. The purpose of the first phase is to provide the planner with the ability to "peek into the future", at least as far as background activities are concerned.

The second, "scheduling" phase is then initiated by resetting the internal clock to the start time of the overall plan. This second phase operates at some multiple of real time and relies on an inter-process communication (IPC) mechanism to exchange data with the ACS simulation. As noted earlier, the ACS simulation is slower than the modeling done within the planner, and can be run at a maximum of 7 times real time. In order to speed up the overall simulation process, we increase speed to 3600 times real time immediately after each turn, and fall back to 7 times real time immediately before starting a new turn.

Another change that is required by the transition to a real-time environment is the ability to communicate with other processes. A widespread mechanism for implementing IPC is the Berkeley socket interface, which is commonly available on both UNIX and PC platforms. Sockets are a low-level interface, but many public domain as well as commercial "wrappers" are available to facilitate the task of the integrator. We followed the lead of our co-workers and chose to use the interface that is already available in the DARTS Shell system, namely the Tool Command Language (Tcl). We refer the reader to Ref. [2] for more details on this topic.

The last step in implementing JIT planning is to put together the real-time capabilities just discussed through an adaptation file that provides the necessary "glue". In our case, we already had at our disposal an activity type that could be used for expressing an observation request. All we had to do was to modify the part of this activity that had to do with slewing. The old slew activity was modified to include the Tcl calls to the DARTS Shell simulator. Fig. 10 displays the new maneuver activity as displayed by Apgen, together with the graphic display of the S/C as simulated by

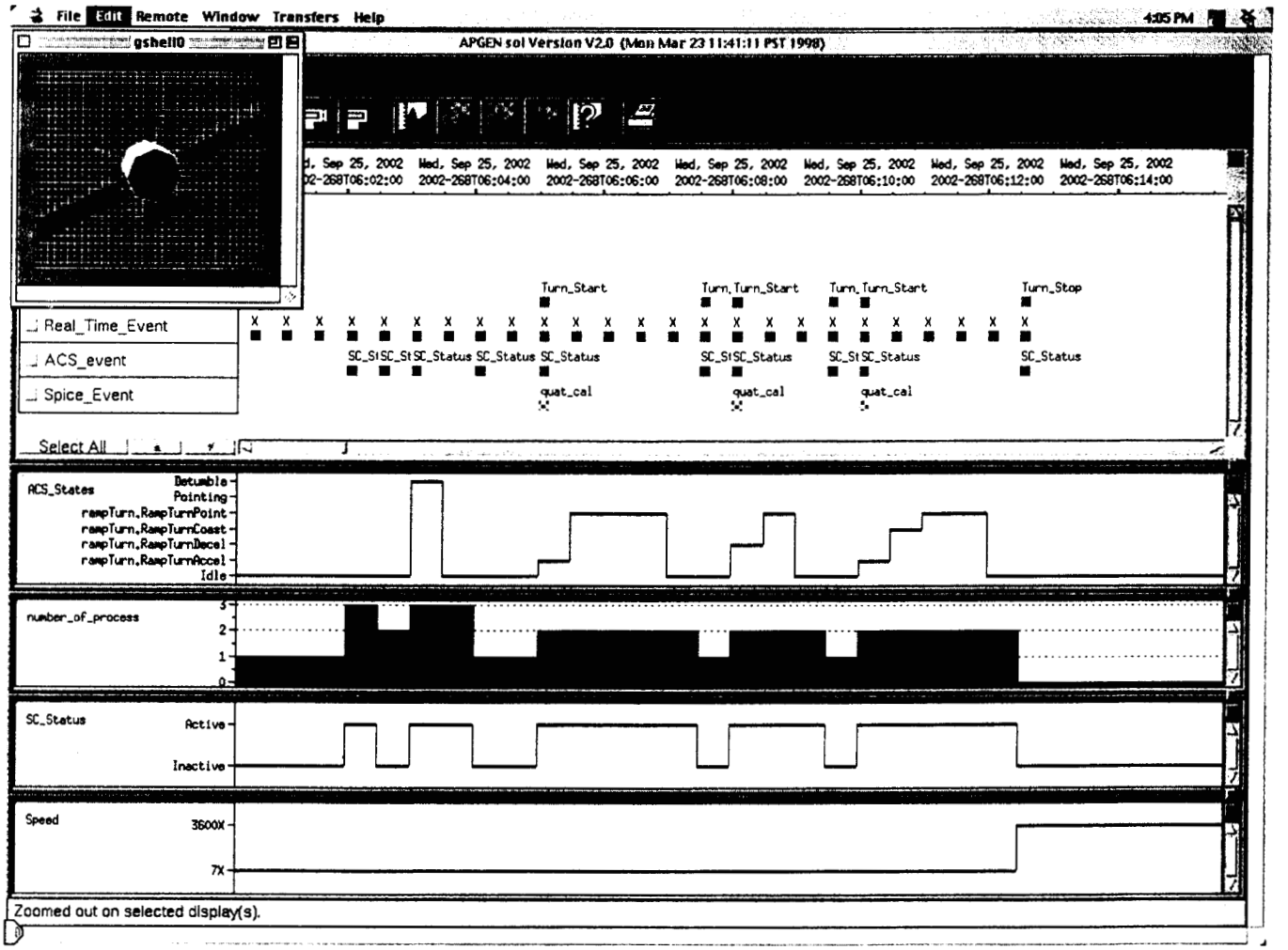

Fig 10: Maneuvers as simulated by Apgen communicating with DARTS Shell at 7 times real time

DARTS Shell.

6. SUMMARY AND FUTURE WORK

The prototype JIT planning system discussed here was implemented using pre-existing planning and simulation tools with very few changes. It took us about 6 months of time to articulate the concept of JIT planning well enough that we could see how we would implement it. It took another 6 months for us to produce the implementation shown here, with one of us (A. K.) working half-time on this project and the other (P. M.) providing consulting advice.

We believe that the work described here exemplifies an integration style that will become more common as space missions require more on-board autonomy. In our discussion of JIT planning, we deliberately avoided the issue of what takes place on the ground vs. what takes place on-board. The use of high-performance, functional simulations of on-board systems in ground data systems should be encouraged, because they provide better fidelity at lower cost than ground-based S/C models. On the flight side, the on-board use of ground-based planning tools such as Apgen would provide flight engineers with ready-to-use, adaptable multi-mission scheduling capabilities that would be costly to implement from scratch.

Looking towards the future, we hope to have a chance of infusing more planning technology into our JIT approach. We have initiated a program of collaboration with our colleagues at the NASA Ames Research Center and are exploring concrete avenues for linking their planning tools [5], [6] into the JPL suite of sequencing tools.

ACKNOWLEDGMENTS

Boris Semenov in Chuck Acton's group provided very helpful guidance in how to link and use the Spice library. Abhi Jain was most helpful in providing us with the DARTS Shell simulation tool, which was developed by him and his group at JPL. This work was performed by the Jet Propulsion Laboratory, California Institute of Technology under contract with the National Aeronautics and Space Administration.

REFERENCES

[1] Jill C. Novak, Felesha Robertson, *Application of Artificial Intelligence to Planning and Scheduling for Operations On-board the Russian Mir Space Station*, Proc. International Workshop on Planning and Scheduling for Space Exploration and Science, p. 28-1, 1997

[2] Jeffrey J. Biesiadecki, David A. Henriquez, and Abhinandan Jain, *A Resuable, Real-Time Spacecraft Dynamic Simulator*, in 6th Digital Avionics Systems Conference, (Irvine, CA), Oct. 1997

[3] Charles H. Acton, Jr , *Ancillary Data Services of NASA's Navigation and Ancillary Information Facility*, Planetary and Space Science, Vol. 44, No.1, pp. 65-70, 1996

[4] Pierre F. Maldague, Adans Y. Ko, Dennis N. Page, and Thomas W. Starbird, *APGEN: A Multi-Mission Semi-Automated Planning Tool*, Proc. International Workshop on Planning and Scheduling for Space Exploration and Science, p. 28-1, 1997

[5] N. Muscettola, HSTS: Integrating planning and scheduling, in Fox, M., and Zweben, M., editors, Intelligent Scheduling. Morgan Kaufmann.

[6] Barney Pell, Erann Gat, Ron Keesing, Nicola Muscettola, and Ben Smith, Robust Periodic Planning and Execution for Autonomous Spacecraft, Proc. International Workshop on Planning and Scheduling for Space Exploration and Science, p. 36-1, 1997

Pierre Maldague *is a software designer in the Ground Data Systems Section of the Jet Propulsion Laboratory. Prior to joining JPL in 1989, he developed mathematical algorithms and applied them to physical systems in a wide variety of settings, including Quantum Mechanics, medical imaging, combustion in laminar flames, Diesel engine design, visualization of Earth based on satellite data, Solids Modeling, CAD/CAM, and image processing. He published a number of papers on theoretical Solid-State Physics, wrote software used in brain surgery, led R & D software teams in several entrepreneurial ventures, and generally has a great time trying to find the right questions to ask. He has a MSEE from Louvain University (Belgium) and a Ph. D. in Physics from MIT.*

Adans Ko *is the Development Manger for the Sequence Subsystem of the Mission Services and Applications Program at the Jet Propulsion Laboratory. During his tenure at JPL, he made contributions to many Flight Missions, including flight software for Voyager and mission planning software for Galileo. He was also responsible for High-Speed Simulation for Cassini, and Uplink Operations as a Team Leader for Cassini. Prior to joining JPL in 1983, he worked on software for statistical analysis of natural resources and built an IT department in Mainland China as Project Manager for Hopewell Holding Co. Ltd. He holds a B.Sc. in Computer Science from Utah State University and an M.B.A. from UCLA..*