Spacecraft Computer Resource Margin Management

Brian T. Larman*
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California

This paper describes the approach being taken by the Galileo project to manage the limited amount of spacecraft computer processing time and memory capacity. The project policy and the rationale behind this approach are presented. The implementation of the policy, as reflected in system requirements and the management methodology, is outlined. The reporting process and how the project responds to a margin over-subscription situation are described. Finally, a summary of the experiences to date, with examples of steps taken as a result of the policy plus an assessment of the impact of the policy upon productivity and system operability, is provided.

Introduction

THE management of the consumption of limited resources is something most of us do regularly in both our professional and personal worlds. Handling of our own finances is probably the most prevelant in our personal world. This same concept is applied routinely to many facets of our technical world. We budget not only funds and time but many other constrained resources as well. In the spacecraft system design world, for example, the "consumption" of mass, power, and volume is routinely managed.

This paper is concerned with the management of the consumption of two relatively new finite and limited resources—spacecraft computer processing time and memory capacity. Until quite recently, for planetary spacecraft these two resources generally fell at opposite ends of the spectrum and, consequently, we did not "manage" them with the same formality we applied to the other critical resources. Computer processing time, at one end of the spectrum, was usually so abundant that the assigned tasks were accomplished within a small fraction of the time available. Thus, processing time was of little concern.

At the other end of the spectrum, memory capacity was so limited that designers looked upon the computer more as special-purpose hardware that was capable of performing only a very limited set of functions rather than as a general-purpose programmable device. Typically, the code was so highly optimized that to attempt to modify it required specialized knowledge (e.g., only a few very specially trained programmers could change it). Thus, reprogramming in flight was simply considered to be too risky except where required to work around failures when there was virtually no other choice.

Even as recently as the current Voyager mission, where some level of memory margin management was exercised, the 4096 16-bit words of the Voyager attitude and articulation control subsystem memory has only approximately 30 unused memory locations scattered throughout the memory. This represents a utilization of over 99%.

The Galileo project orbiter, with 18 microcomputers and the equivalent of 360 K 8-bit bytes of memory contained within two major engineering subsystems and eight science instruments, has defined a project policy requiring that the key onboard computer system resources be managed in a very rigorous and visible manner. This paper describes that policy and the rationale behind it. It then discusses the methodology used to implement that policy and summarizes some of the steps already taken as a result of the resource management process. To place the margin management methodology into the proper perspective, an overview of the Galileo mission and spacecraft system is provided.

The Galileo Mission and Applicable Spacecraft Systems

The Galileo spacecraft consists of two vehicles: a probe and an orbiter. The two vehicles will make the journey from earth to the vicinity of Jupiter while they are joined together. During the interplanetary cruise phase, the probe will remain unpowered except for brief periods of system checkout. Approximately 150 days prior to encountering Jupiter, the probe will be powered, targeted for Jupiter, and then released. It will collect data as it enters the Jovian atmosphere and transmit this data to the orbiter for relay to earth.

After completing the relay link task, the orbiter will execute a maneuver to place it into an elliptical orbit about Jupiter. From this vantage point, this dual-spin vehicle will obtain both remote-sensing observations (e.g., pictures) of Jupiter and several of its satellites and in situ fields-and-particles (e.g., electromagnetic, plasma, and particle) observations.

Radio science observations are also planned via the orbiter's *S*- and *X*-band telecommunications subsystems. Within the orbiter, the two major computer-based engineering subsystems that are the topic of the remainder of this paper are the attitude and articulation control subsystem (AACS) and the command data subsystem (CDS). These two subsystems are described below. Software development for science instruments is described in Ref. 1.

AACS

The prime functions of the AACS are to autonomously determine and control the attitude of this dual-spin vehicle in inertial space and to provide a stable platform that will allow precise, two degree-of-freedom pointing of the remote-sensing instruments. The AACS also controls the spun section which allows 4π-steradian field of view for the fields-and-particles instruments.

The AACS contains an imbedded redundant pair of ATAC-16M computer systems with 31K (16-bit words) of random-access memory (RAM) and 1K (16-bit words) of read-only memory (ROM) each.

A detailed discussion of both the AACS and the control analysis solutions may be found in Ref. 2.

Presented as Paper 81-2123 at the AIAA Computers in Aerospace III Conference, San Diego, Calif., Oct. 21-23, 1981; submitted Oct. 21, 1981; revision received June 21, 1982. This paper is declared a work of the U.S. Government and therefore is in the public domain.

*Member, Technical Staff, Spacecraft Systems Engineering Section, Systems Division. Member AIAA.

CDS

The prime functions of the CDS are to coordinate all spacecraft instrument observation operations, engineering supporting operations, system-level fault protection, and onboard data collection and formating, plus decoding and distribution of command data received from earth.

The CDS contains six RCA 1802 8-bit microcomputers and a total of 176K 8-bit bytes of RAM. These computers are configured into a distributed architecture of two virtually redundant sets. Each set consists of one 1802 and 32K of RAM, called a high-level module (HLM), plus two 1802s and 16K of RAM each, called low-level modules (LLM). Both sets, or "strings," have common access through an intercommunications data bus to four additional memories with a total of 48K of RAM for buffer storage.

A detailed discussion of the CDS can be found in Ref. 3.

Galileo Project Computer Resource Management Policy

The project policy may be paraphrased as follows: the amount of onboard computer processor time and memory capacity estimated to be necessary to implement all requirements shall not exceed an amount specified at each major milestone in the development process. Figure 1 indicates the allowable resource utilization vs development milestones. The key values are 55% utilization at the preliminary requirements approval stage, 75% utilization at launch, and 85% utilization at Jupiter orbit insertion (approximate beginning of the orbital operations phase of the mission). As will be explained in a later section on AACS, the limit for that subsystem recently has been raised to 85% at launch.

Rationale behind the Policy

The justification for this policy is derived from our own previous planetary flight experience and a strong recommendation from the Space Shuttle flight software management staff. Additional justification is derived from industry experience[4,5] that identifies the effect upon programmer productivity, program maintainability, and program understandability of attempts to program a computer in which the memory capacity and/or processor time limit is approached. Basically, the main theme of all of these factors is: as the limit of the resource is approached, the time required to develop a reliable working program increases asymtotically. Figure 2, taken from Ref. 4, is based upon considerable practical experience and shows that the knee of the curve of productivity vs computer resource utilization is around the 75% utilization point.

The Galileo project expects that reprogramming of both AACS and CDS software will be necessary after initial delivery (i.e., during spacecraft system testing) and after launch. The project has therefore established a companion policy that the flight and ground systems be designed to facilitate this reprogramming activity. There are many reasons for this expectation. In the case of the AACS, providing autonomous attitude determination and control for a dual-spin structure of the magnitude and complexity of the Galileo spacecraft is a state-of-the-art process. Furthermore, the response of this configuration to actual flight conditions in the weightless vacuum of space can only be estimated. Though mathematical models have been developed, neither actual experience nor test facilities are available to completely verify their accuracy. Several other factors contribute to the expectation that reprogramming of the AACS will be necessary. Examples of these factors are:

1) Hardware procurement and software development are being done basically in parallel. Software changes to accommodate hardware characteristics which are either different than specified or which exhibit unexpected idiosyncracies may be necessary.

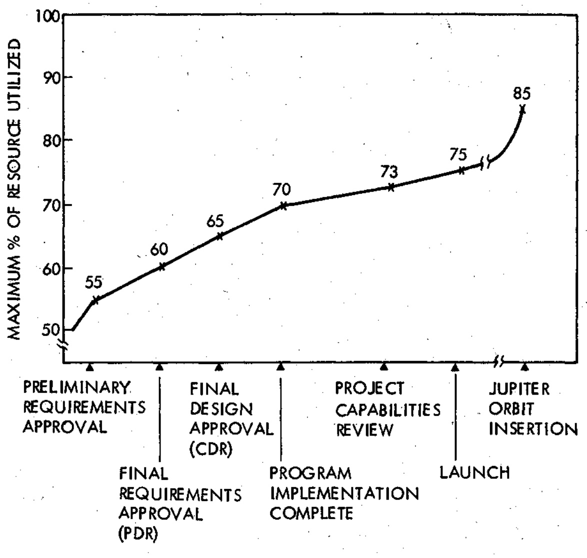

Fig. 1 Project-allowed resource utilization.

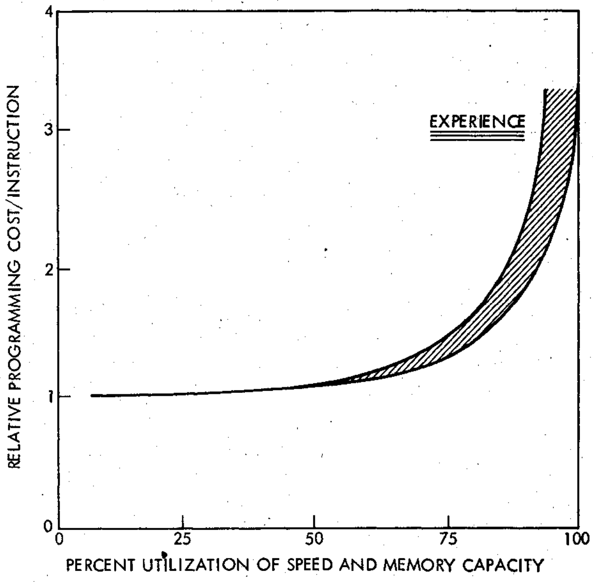

Fig. 2 Productivity vs resource utilization.

2) Spacecraft mass properties changes (e.g., fabricate stiffness or a mass element differs from design estimates) ma necessitate some attitude control algorithm modifications The software that implements these algorithms must chang accordingly.

3) Modifications to spacecraft system performance or faul protection requirements may affect software design an implementation.

4) Hardware performance degradation or failure in fligh may force flight software modifications.

Some of the examples above also apply to the CDS. Othe factors that may precipitate CDS software reprogrammin are:

1) The effect of a partial failure or degraded operation o an instrument may be compensated for by a modification t CDS software.

2) Some onboard fault protection algorithms will b modified as mission phases vary (e.g., from the very quiescen cruise phase to the extremely critical Jupiter orbit insertio activity).

3) It may become necessary to change the algorithms use to coordinate operation of and data collection by scienc instruments to compensate for unexpected environmenta

conditions or other circumstances encountered by the orbiter once it is in orbit and is actually executing programmed activities.

4) The allocations of functions between spacecraft and ground functions may change with mission phase or as flight operations evolve.

Thus, reprogramming is expected to be a fact of life. The policy was established to insure that adequate computer resources would be available to make that reprogramming not only feasible but cost effective. However, simply defining a policy does not insure that the intent will be met. The policy must be implemented. The next section describes the implementation approach.

Defining Margins

Clearly, when the software is delivered, the amount of memory required is easily determined by inspection. Processor time can be determined through measurements using tests constructed specifically for that purpose; however, during the early development stages, estimating these resource requirements is much more difficult.

Estimating processor time requirements, particularly for a distributed system, is an especially enigmatic task. Several techniques have been tried by persons who have attempted to develop more reliable and accurate methods of estimating and measuring resource utilization. Different techniques are appropriate at different development stages. The more useful of these are outlined below.

Early Requirements Development Stage

Once the specific machine is selected, a general software architecture that best supports the overall set of allocated functions is established. The resources required to perform similar functions on previous missions were used as a basis for estimating the Galileo project needs, taking into account differences in machine, word size, instruction set, etc. For example, a function which required n 18-bit words in the Voyager equivalent of the CDS was assumed to require $6n$ 8-bit bytes of Galileo CDS memory. The rationale for this factor was that $3n$ compensates for the word size differences and an additional average of $3n$ is necessary due to differences in instruction set and coding efficiency. This is because the CDS's RCA 1802 machine is significantly different from the machine which had been specifically designed and tailored for the Viking and Voyager missions. The required use of a high-order language was also expected to introduce some inefficiency over the highly optimized assembly language code used in this subsystem on prior projects. Benchmark examples were constructed to validate this scaling factor.

A similar scaling process was used in estimating AACS resources at this stage due to the requirement that a high-order language be used and because control algorithms were expected to be significantly more complex. This approach provided a means for estimating memory sizing. Estimates of processor time based upon a similar extrapolation, taking into account differences in instruction types, machine characteristics, etc., were also done. Estimates of functions which could not be extrapolated in this manner were based simply on "best guesses."

Requirements Finalization/Preliminary Design Stage

By this stage of development, a much better understanding of the total scope of the requirements existed. In addition to a relatively firm architectural design, prototype designs of a broad representative set of modules were evolving. For both AACS and CDS, structured design language versions of some modules had been developed and some of these had been coded. The average number of words of memory per design language statement could be used as a rough guide to estimating memory requirements for similar functions once the design language statements for these were developed. Since AACS control algorithms must be validated via simulations which interface with mathematical models of the spacecraft structure and environment, FORTRAN or even HAL/S (the AACS high-order language) versions of these became available for sizing analysis as they were being developed. To develop timing estimates, the worst-case path through each of the prototype code modules was determined and the processor time to complete that path was computed manually. This tended to produce very pessimistic results when totals were accumulated since it is highly unlikely that, for any given period, the worst-case path would be traversed by every module. A reasonable method of dealing with this problem is yet to be defined.

Design/Implementation Stage

This stage has just begun. As actual code modules are developed, the number of memory locations are counted and processor time either is measured during module testing on the development system or is determined manually. These "real numbers" are used to improve the accuracy of the estimates for the other modules yet to be coded to updating the memory words per design language statement algorithms.

Optimization or "Scrubbing" Stage

It is expected that design and code "scrubbing" will be necessary. Actual code and test measurements will be used to determine accurately the resource utilization data. "Scrubbing" is the process of revising control algorithms and flight code to accomplish the same tasks at equal or better performance with more efficient utilization of computer resources. Using these techniques, a measure of the resources required to implement the requirements can be obtained and compared against the total available. This allows us to estimate the amount of margin available at any given time in the development cycle with emphasis on improving the estimation accuracy as the design matures.

Implementing the Policy

Overview

The policy is implemented through the combination of a management methodology and a set of system-level requirements levied on the design of the computer-based spacecraft subsystems. Examples of the latter are:

1) Each computer system must measure and report via telemetry the amount of processor time utilized per unit time.

2) Ground operations software must provide predictions of the amount of spacecraft memory required to contain the parameters required to implement each set of observations.

3) A set of criteria has been defined to be utilized in deciding if a set of spacecraft observation functions should be expanded via flight or ground software. A memory sizing tradeoff is a major element of these criteria.

The major contributor to implementation of the policy, however, is the management methodology. There is heavy use of prototype coding to prove feasibility and provide a measure of computer resource utilization. This is an application of the "plan to throw one away" or "plan to do it twice" concept popularized by Brooks.[6] A specified period of time is provided in the schedule for AACS control algorithm scrubbing and both AACS and CDS requirements and code scrubbing. Memory sizing vs processor time tradeoffs are a major influence. At the more detailed level, slightly different approaches toward implementing the policy for AACS and CDS are being taken.

AACS

As noted above, the AACS development is state-of-the-art. Due to hard launch date constraints, the completion date is fixed and cannot slip (i.e., the spacecraft must be launched within the approximately 10 day "launch window," or the entire mission will be delayed by at least 13 months).

The software development task is further complicated by the fact that the project schedule forces a relatively large overlap among spacecraft configuration definition, control algorithm design, and the software development efforts. Thus, the software implementation is well underway before the spacecraft configuration (and, as a result, the algorithm design) is frozen. Thus, a dilemma is to be faced: if, at any point during the development, the estimate of memory capacity or processor time required to implement a particular algorithm significantly exceeds the amount allocated, a choice must be made between designating how much of the limited available manpower should be diverted to solving that problem vs allowing all of the staff to continue working the other "unknowns" in an attempt to produce a working (albeit oversubscribed) system. The procedure for choosing between the options is as follows: if the oversubscription is less than 100% of the available resource, the first step is to determine if the various known approaches to reducing it are feasible. If they are feasible, then the decision is made to proceed with the rest of the development. If they are not feasible or if the oversubscription is greater than 100%, manpower must be diverted until a feasible solution is defined. Creating a working program is considered to be the first priority.

The Galileo project has required that the AACS flight code modules be designed and coded in the high-order language HAL/S. This language was originally developed for the NASA Space Shuttle flight computers. A HAL/S compiler was developed for the ATAC-16M machine and, while the processor timing efficiency specification was met, the memory utilization specification was not met. Therefore, the Galileo project has authorized an increase in the margin limit for this subsystem at launch from 75 to 85%. The project believes that, after actual AACS flight applications experience with HAL/S is gained, additional specific compiler optimizing actions will be feasible, thereby improving efficiency and allowing some of this margin to be recouped.

CDS

In the case of the CDS, a more serial process in the traditional structured approach of requirements, followed by design, and then coding and testing, is being taken. However, as noted above, there is considerable emphasis upon prototype coding of key design features very early in the process. Development is phased in a structured, top-down manner. Time for code scrubbing is provided.

Because of the influence of the distributed architecture of the CDS upon all interfacing subsystems and instruments and the fact that the majority of spacecraft system-level functions are implemented in this subsystem, the management of system-allocated CDS resources at the major function level is performed by the engineer responsible for that major function. CDS was also originally required to use HAL/S. Since were were unable to obtain an adequately efficient 1802 HAL/S compiler, a macro-based assembler approach in concert with a set of design and coding practices and constraints was substituted.

Reporting Margin Status

Margin reports are provided monthly by the implementing organizations. The measurement and reporting system is structured to indicate current status with increasing levels of detail to provide visibility to the level required for any specific situation.

Table 1 is the AACS memory-sizing margin summary report which is provided to upper-level Galileo project management monthly. It clearly indicates whether or not the total amount of memory estimated to be required to perform the AACS functions under the current design is above or below the value allowed by the policy as of that date. Table 2 is the next level of detail indicating the status of the major functions. Table 3 is a section of one page of the lowest level of detail reported. This is the working level and clearly indicates the current status and change activity down to the individual task. Similar reports are provided for processor time utilization. Reports for CDS resource utilization, though formated somewhat differently, are also generated.

Response to Allocation Oversubscription

Each higher-level report is derived directly from the lower-level data. Thus, management at any level has available consistent data at whatever level of detail is necessary to

Table 1 AACS memory margin status summary, July 20, 1981

Total HAL/S statements	3,389
Total ATAC code words	23,471
Total ATAC data words	6,701
Grand total	30,172
Change since last report (06/19/81)	−207
Uncertainty (3σ)	1,126
Percentage used = 92.08%	
Margin = 7.91%	

Table 2 AACS memory utilization vs major function

Function type	No.	HAL/S State-ments	Code words	Data words	Change
Executive	2	300	4000	1700	0
Attitude determination	10	1062	6838	2251	0
Attitude/scan control	14	1596	8153	1661	−207
Fault protection	4	250	2124	135	0
Command/telemetry	4	161	485	936	0
Utilities	11	20	1871	18	0

Table 3 Memory utilization by code module

Function	HAL/S statements	Code words	Data words	Total words	+ or − words	Actual code?	Scrub code?	Change
Acquire	75	307	79	386	100	Yes	No	0
Arrays	0	153	0	153	50	Yes	No	0
$A_{sin} \cdot A_{cos}$	0	99	0	99	0	Yes	Yes	0
A_{tan_2}	0	119	0	119	0	Yes	Yes	0
AXLDVC	90	288	93	381	50	Yes	No	0
Batch star identification	425	3226	1394	4620	200	Yes	Yes	0
Check sum	0	50	10	60	20	Yes	No	0
Command processor	25	76	587	663	100	Yes	No	0
Configuration management	75	450	50	500	120	No	No	0
Coordinate transfer	150	800	100	900	200	Yes	No	0
Deltv. estimate	10	60	15	75	25	No	No	0
Encode rate estimate	12	84	28	112	50	Yes	No	0

thoroughly evaluate the margin situation. When a report indicates that a margin limit has been exceeded, the immediate response by system engineers and managers is to analyze the reported cause of the oversubscription and determine if it is transient (e.g., first cut, highly inefficient design just to prove feasibility) or if it represents a major lien (e.g., an efficient design which is probably the best we can expect to implement to meet the requirements). In the former case, the usual response is to watch the offending area as the design evolves and if it has not improved within two months, additional, more drastic steps are taken. In the latter case, any combination of one or more of the following steps may be taken to eliminate an oversubscription problem:

1) Algorithm scrubbing: the design of the algorithm is analyzed in search of more efficient ways of meeting the requirements.

2) Design scrubbing: the software module design is reviewed to determine alternative approaches to implementing the algorithm which will reduce the oversubscription problem.

3) Code scrubbing: if the oversubscription appears after a module is coded, alternate coding techniques are evaluated.

4) Subsystem architecture and requirements tradeoffs: the overall subsystem software, and even hardware if applicable, is analyzed to determine if changes at that level can reduce the oversubscription without jeopardizing other modules.

5) System requirements tradeoff: orbiter system design and functional and performance requirements are reviewed to determine if requirements may be deleted, waived for this module, or eased (particularly performance requirements) to reduce the oversubscription. Another alternative may be to reallocate the function to a different computer or to the ground.

6) Utilization limit increase: a decision can be made to allow the oversubscription to remain (i.e., do nothing about problem and accept the risk).

The key is that a visible, conscious, project-level decision is made. As with any management task, particularly one which is new and somewhat controversial, the tendency to overmanage must be avoided. As both technical and managerial personnel better understand what must be measured, how to improve estimating techniques, and how to express the uncertainties, responses to unexpected events must be tempered with rational and carefully considered actions so that all parties gain confidence that the process is yielding meaningful and useful results. If managers react harshly to a report of a sudden increase in resource utilization, the technical people tend more toward "creative accounting" aimed at reducing the appearance of limit overruns in their reports. This may give the appearance that all is progressing well, but it hides the real oversubscription issues early in the process. This is precisely what the process is designed to uncover. Therefore, techniques to clearly depict the resource utilization picture and to deal with oversubscriptions in a realistic, intelligent manner are needed. To do so in a manner which does not seriously penalize any of the team members but still works effectively to facilitate accomplishment of the best possible mission within the available resources is the real trick. This process has been working within the Galileo project for some time. The section below provides some examples of the decisions made so far as a result of the margin management process.

Examples of Some Steps Taken

Two examples of actions taken with the AACS are:

1) In mid-1979, the estimate of the amount of memory required to implement the AACS functions as understood at that point was well in excess of the 50% allowed by the project policy. As a result of the analysis, 12 K of memory was added to the hardware complement for each processor. Since the latest (as of this writing) margin report indicates that both memory capacity and procesor time utilization are very close to the allowable upper limit, this was obviously a very wise decision.

2) Earlier this year, the AACS scan commander algorithm was completed. When the software designers attempted to implement it, they estimated that it would cause the processor time utilization to increase to approximately 98% of total. This was such a large departure from precompletion estimates that a significant effort was begun immediately to investigate methods of reducing this oversubscription. By moving some functions to lower frequency rate groups, the overall utilization estimate was reduced to approximately 85%. Still more optimization is required here, but this is a good example of the process in action. The designers are confident that the same technique can be applied to other existing algorithms during the scheduled scrubbing period.

Concerning the CDS, two major modifications have been made as a direct result of the margin management process. These are:

1) Very early in the development process, the CDS architecture consisted of three HLMs with 16 K of RAM each. Early estimates indicated that there would be inadequate memory capacity available. Analysis indicated that the requirements could be better met with two HLMs with 32 K of RAM each. Therefore, this hardware and architectural modification was made.

2) The project had a requirement that both CDS and AACS software be developed using the high-order language HAL/S. The compiler for the CDS's 1802 machine did not meet the efficiency specifications. Analysis indicated that, with the use of HAL/S and the current design, both HLM memory capacity and processor time would be severely oversubscribed. After considerable effort and soul-searching, the project waived the HAL/S requirement. The code now will be written using a combination of structured macros and assembly language. At this same time, some software architecture modifications were made and some subfunctions were reallocated from the HLM to the LLMs. These steps brought the estimated utilization back to well within the required level.

One other example serves to highlight another facet of the influence of the margin management methodology upon the overall system development. The mission sequence system (MSS) is that collection of ground-based software and procedures that is required to generate and validate the detailed sequences of spacecraft instrument and subsystem activities necessary to acquire and transmit to earth the science observation data. This is a prime purpose of the mission. To perform this task, the MSS must utilize the limited onboard resources. One set of these resources is CDS processor time and memory. To facilitate an end-to-end system design, a portion of these CDS resources was allocated recently to the MSS engineer so that flight/ground resource utilization tradeoff could be considered in the design of the MSS architecture and in the operational strategy development. For example, if a spacecraft function is to be repeated may times, it may be appropriate to use a portion of the onboard memory to contain a subroutine to perform this function and to load a table of parameters in the CDS memory. On the other hand, if it is a one-time-only function, a simple table of events to be executed once only and then discarded may be a more efficient utilization of CDS memory. The strategy and overall MSS architecture will be dependent in part upon how much of the CDS resources are available to it. Therefore, an allocation to the MSS was made and the responsible engineer is now required to manage that portion of the CDS resources within the same project policy as the other onboard subsystems.

Conclusions

Because of the uniqueness of the Galileo spacecraft system, the development environment, and the mission itself, the capability to reprogram the onboard computers is essential. It is also essential that this reprogramming activity be cost effective and timely, and produce highly reliable, robust code.

One of the prime factors in achieving this requirement is that an adequate amount of both memory capacity and computer processing time be available to facilitate it. The reporting process provides engineers and managers with the information necessary to make informed intelligent decisions. The visibility and sensitivity at all levels that are a direct result of the project's computer margin management policy, and the steps taken to implement it, will make the reprogramming task feasible. Indeed, it has already had a major influence on subsystem hardware and software architecture and design and continues to be a prime influence in the ground-based mission sequence system development effort. As spacecraft achitectures become more computer-based, instruments and missions become more sophisticated, and more emphasis is placed upon reprogramming spacecraft computers in flight, rigorous management of the limited computer resources throughout the software development and maintenance phases is essential.

Acknowledgment

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

References

[1]Barry, R.C. and Reifer, D.J., "Galileo Flight Software Management, The Science Instruments," *Proceedings of IEEE COMSPAC 80,* Chicago, Ill., Oct. 1980, pp. 684-690.

[2]Rasmussen, R.D. and Brown, T.K., "Attitude and Articulation Control Solutions for Project Galileo," Paper presented at American Astronautical Society, Annual Rocky Mountain Guidance and Control Conference, Keystone, Colo., Feb. 1980.

[3]Kohl, W.H., "Galileo Command and Data Subsystem," *Proceedings of Industry/Space Division NASA Conference and Workshop on Mission Assurance,* Los Angeles, Calif., Apr.-May 1980, pp. 326-333.

[4]Boehm, B.W., "Software and its Impact: A quantitative Assessment," *Datamation,* May 1973.

[5]"Weapons System Software Development. MIL-STD-1679 (Navy), Dec. 1, 1978.

[6]Brooks, F.P., *The Mythical Manmonth, Essays on Software Engineering,* Addison-Wesley, Reading, Mass., Dec. 1979, pp. 115-118.