On-Board Maintenance for Long-Life Systems

Ann T. Tai
IA Tech, Inc.
10501 Kinnard Avenue
Los Angeles, CA 90024
a.t.tai@ieee.org

Leon Alkalai
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109
Leon.Alkalai@jpl.nasa.gov

Abstract

Due to the low power, low cost, high reliability and high performance goals, the traditional approaches to fault-tolerant, ultra-reliable systems that rely on custom-built hardware and extensive component/subsystem replication will not be feasible for the new-generation spaceborne computing systems. In this paper, we present a new concept called "on-board maintenance." We classify on-board maintenance to three categories: Preventive maintenance, perfective maintenance, and corrective maintenance. For each type, we present the definition and propose some approaches to its realization.

1 Introduction

Although long-life systems used to be defined as the type of systems that are never maintained, the new-generation space exploration missions aimed at both high-reliability and low-cost necessitate on-board maintainability [1]. As the technologies in telecommunications, distributed system architecture, fault-tolerant computing and software engineering advance rapidly, on-board maintenance has become achievable. Similar to the conventional notion of system maintenance, on-board maintenance collectively refers to preservation or improvement, during its operational life, of a system's ability to deliver a service complying with mission requirements. Accordingly, on-board maintenance can be classified into the categories of perfective maintenance, preventive maintenance and corrective maintenance.

The remainder of the paper is organized as follows. Section 2 provides background information by describing a new-generation spaceborne computing system. Section 3 describes on-board maintenance in terms of three categories, followed by Section 4 which discusses the relationships between different types of on-board maintenance and possible methodology coordination. The concluding section provides a summary.

2 Background

X2000 is a new-generation space technology program that will provide an engineering model to multiple long-life deep-space missions [2, 1]. Currently, five missions are designated to be the recipients of the X2000 technology: Europa Orbiter, Pluto-Kuiper Express, Solar Probe, Mars Sample Return and DS4/Champollion (also known as Comet Sample Return). One of the major challenges the X2000 program exhibits to us is the requirement diversity among the five missions, which demand a computation power from a single processor string to multiple strings, a throughput range from under 20 MIPs to over 100 MIPS, and a mass memory size from 100 Mbytes to 1.5 Gbytes. Therefore, the X2000's computing system architecture must be scalable and distributed in order to accommodate a broad spectrum of requirements. As far as the avionics concern, the X2000 is aimed at further advancing the packaging technologies initiated by the New Millennium Deep Space One (NMP DS1) program [3, 4]. The NMP DS1 developed an architecture which consists of a RAD-6000 processor multi-chip-module (MCM), a local memory MCM, a non-volatile mass memory MCM, and an I/O MCM. The X2000 architecture has taken a further step toward miniaturization, in which each processor string consists of an processor slice integrated with I/O interfaces, a local memory slice, and one to four non-volatile mass memory. Moreover, the X2000 architecture has been extended through employing multiple processor strings connected by redundant buses, namely, IEEE 1394 and I2C, to enhance mission reliability and performance [5]. As the use of standard buses enables the X2000 architecture to be scalable, the system can accommodate from one to multiple processor strings. An instance of a two-string configuration of the X2000 architecture is depicted in Figure 1.

A main feature of the X2000 system architecture

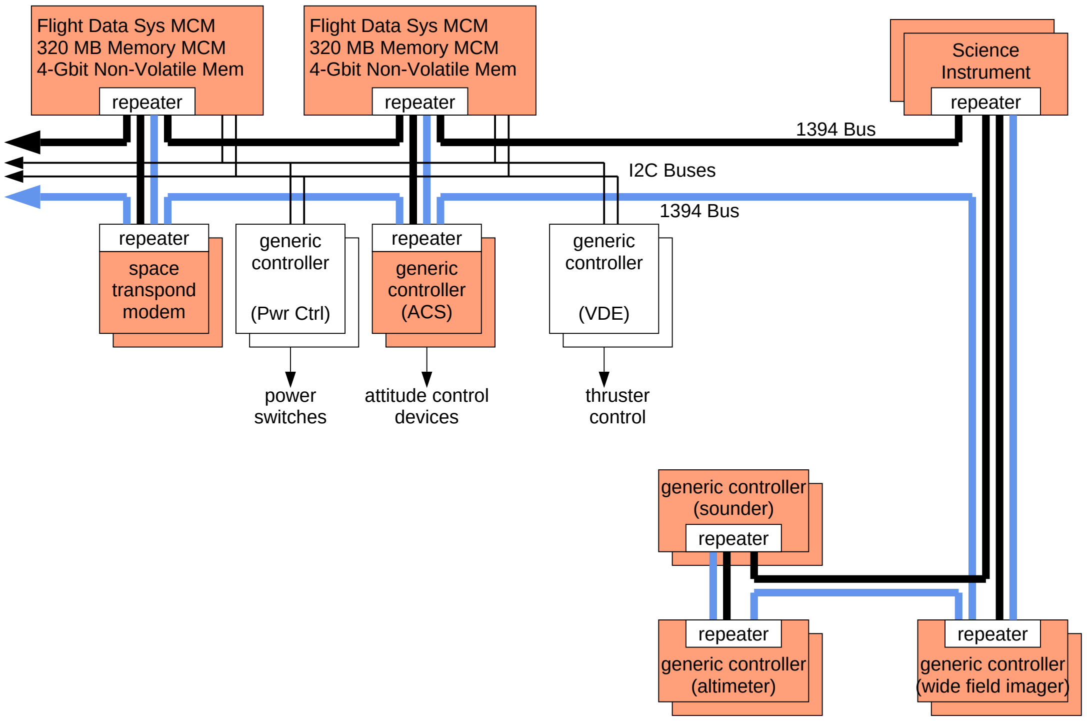

Figure 1: X2000 System Architecture

is the I/O cross-strapping for the processor strings that is implemented using the standard interfaces of IEEE 1394 and I2C. This feature permits the workload that comprises the spacecraft and science functions to be shared by and migrated between processor strings in an efficient manner. Therefore, the I/O cross-strapping plays an important role in enabling adaptive utilization of system resource redundancies, a cost-effective way to the realization of dependability and performance enhancement. Among other things, on-board maintenance schemes have been proposed to take advantage of this feature. For example, on-board preventive maintenance can be realized as follows [6]: During less critical mission phases such as cruise phases, the processor strings will be scheduled on and off duty periodically, servicing the mission on a rotation basis. The benefit from this approach is two-fold: 1) a significant saving for the limited power on-board, and 2) periodic rejuvenation for both hardware and software of the processor strings.

3 On-Board Maintenance: Definitions and Approaches

3.1 Definitions

Although the general concept of on-board maintenance is similar to the conventional notion of system maintenance [7], due to that on-board maintenance is conducted during a space exploration mission with limited ground support, the definitions of preventive maintenance, perfective maintenance and corrective maintenance differ from those in the context of system/software engineering. In particular, on-board preventive maintenance and corrective maintenance focus on system errors. On the other hand, a major objective of perfective maintenance is to remove the underlying faults that cause error conditions and to improve system's fault tolerance capability. Namely,

Preventive maintenance removes or minimizes potential *error conditions* that accrue over a system's operational life before they produce symptoms.

Perfective maintenance improves or upgrades system's performance, dependability and fault tolerance capabilities in order to adapt a system to the evolving mission/system requirements.

Corrective maintenance rectifies *errors* (including those introduced by perfective maintenance) and/or mitigates their effects on system operation after the errors are detected.

3.2 Preventive Maintenance

With respect to long-life space-exploration missions, reliability implies a system's continuous operation up to 15 years and availability is the system's readiness to service a mission in an unknown deep-space, radiation intense environment.

On-board preventive maintenance generally refers to the actions taken place during a mission for eliminating or minimizing potential error conditions that accrue over the operational life of a spaceborne system. Although the concept is analogous to that of "software rejuvenation" which has received an appreciable amount of attention in the past few years [8, 9, 10], on-board preventive maintenance concerns both hardware and software, and must be realized in a manner which keeps the maintenance-caused system unavailability minimal. Specifically, from software perspective, aging phenomenon such as memory leakage and data corruption can be removed via program reinitialization which cleans up a system's internal state (complete age reversal); from hardware perspective, the effects of electronmigration (the driving force of circuit failures), which occurs in microelectronic devices when current density is high, can be reduced through structural/thermal relaxation during a power-off period [11, 12] (partial age reversal). Accordingly, the procedure of on-board preventive maintenance for the X2000 spaceborne computing system typically involves 1) stopping the running software and host hardware system, and 2) rebooting the system and restarting software execution after a scheduled time interval. To minimize maintenance-caused system unavailability, we exploit nondedicated system redundancy. An instance of nondedicated redundancy in the X2000 computing system is the following: During a critical mission phase which demands a full computation power (such as the Encountering Phase in the 15-year long Pluto-Kuiper Express mission, see Figure 2), all the processor strings are scheduled to jointly perform the spacecraft and scientific functions, while only a subset of the strings is mandated to be in service during less-critical mission phases such as a cruise phase. Hence, individual processor strings can rotate between on-duty and off-duty shifts based on a scheduled time interval (see Figure 3), we call it a *duty period*, for preventive maintenance throughout the mission except during the phase(s) requiring a full computation power. In this manner, on-board preventive maintenance practically has no negative effect on system availability.

Our initial model-based study demonstrates the feasibility of on-board preventive maintenance [6]. Recently, we extend our basic model to facilitate phased-mission analyses. Inspired by the results of our earlier study which revealed that optimal duty period is operational environment dependent [6], we utilize the extended model and the mission profiles of Pluto-Kuiper Express and DS4/Champollion to investigate into the influence of adjusting duty period to mission phases on reliability gain [13]. The evaluation results confirm a potential for significant gains in mission reliability from on-board preventive maintenance and provide to us useful insights about the collective effect of age-dependent failure behavior, residual mission life, risk of unsuccessful maintenance and maintenance frequency on mission reliability. In particular, the model-based analyses reveal that phased-adjusted on-board preventive maintenance (in terms of maintenance frequency) will lead to further reliability gain.

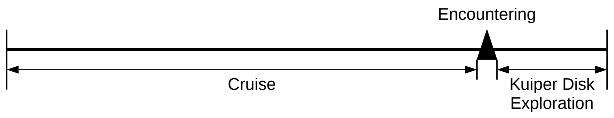

Figure 2: Mission Phases of Pluto-Kuiper Express

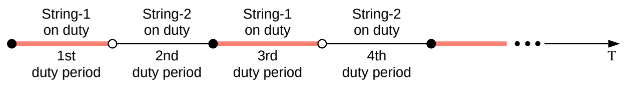

Figure 3: Duty-Switching Sequence

3.3 Perfective Maintenance

In order to simultaneously achieve the low power, low cost, high reliability and high performance goals, the new-generation spaceborne systems can no longer use the traditional approaches to fault tolerance that rely on custom-built hardware and extensive component replication. Among other challenges, spaceborne computing systems for long-life deep-space missions are anticipated to have the on-board capability to accomplish enhancement of dependability, performance and functionality with minimal ground support, referred to as perfective maintenance (also known as evolvability [1]).

Related concepts include hardware reconfigurability and software upgradability [2]. Hardware reconfigurability refers to the on-board ability to create new electronic functions as conditions change during a mission. The creation of new functions will be accomplished using reconfigurable logical gates coupled with on-board (possibly genetic) algorithms.

Software upgradability provides us with the following benefits:

1) It significantly reduces the time taken to complete and install the first version.

2) A particular software module (instead of the entire flight software) can be uploaded without the costly interruption of missions normal functions.

3) Upgradable software permits a spaceborne system, during its mission's long life span, to keep pace with the latest software technologies for better performance, fault tolerance and functionality, instead of being constrained by those available prior to mission launch.

Categories of software upgrade include the following:

1. Addition of a new scientific/spacecraft function,
2. Performance enhancement of an existing function,
3. Fault tolerance enhancement for an existing function,
4. Precision improvement for an existing function, and design/coding fault removal.

3.4 Corrective Maintenance

With an evolvable system, to maintain the consistency between old and new versions with respect to their fault detection and recovery functions in order to avoid gaps in fault tolerance protection is crucial [14]. NASA experienced such a gap on April 10, 1981, when a timely synchronization check was omitted after the addition of an alternate reentry program. As a result, the first flight of the US space shuttle program was aborted 19 minutes before launch. Therefore, among other research opportunities which the new generation of spaceborne system exhibits to us, a particular challenge is to develop innovative methods that assure reliable system evolution, especially software upgrading. The approach to the assurance are two-tiered:

1) To thoroughly verify and validate software modules that are modified or implemented for upgrading prior to uplinking, and

2) To minimize the adverse effects from upgrading due to residual faults in upgraded software modules.

While the former primarily concerns improved software engineering practice, the latter corresponds to on-board corrective maintenance. On-board corrective maintenance can be conducted between upgrades through the use of fault tolerance techniques.

To simultaneously meet the low-power, low-weight, low-cost and high-reliability criteria, we can achieve on-board maintenance by exploiting nondedicated resource redundancies. These include the processing and storage elements in a parallel/distributed system and the software versions uploaded to a spaceborne system through a series of upgrading.

Nondedicated resource redundancy can be utilized to avoid system failures caused by on-board software upgrading. An earlier, relatively more mature version can be used as a backup module. When an error caused by a residual software defect is detected in the updated version, the backup module can be used to restart the failed task based on checkpointing. When an error is detected in a newly added function, which means the corresponding backup module is unavailable, forward recovery techniques can be used.

Checkpointing, the most commonly used fault tolerance mechanism, can be adapted for corrective maintenance. In particular, the determination for error recovery and isolation mechanisms (in a distributed computing environment) associated with checkpointing is driven by the type of upgrading a software module has undergone. Figure 4 depicts a possible mapping between upgrading types and recovery mechanisms, where two kinds of recovery scheme are employed, namely, backward recovery and forward recovery.

Imprecise computation [15] is a forward recovery means. It reduces the adverse effect of timing errors (due to slow numerical convergence, for example) and achieves graceful degradation by giving the user an approximate result of acceptable quality when the system is unable to produce the exact result in time. The imprecise computation technique uses this strategy and divides tasks into mandatory and optional subtasks. If the task is terminated before completion, the intermediate result can be used as long as the mandatory subtask completes. With respect to our application, imprecise computation can be employed for forward recovery of upgrading-related (value or timing) errors, given that the software module in question can be decomposed into and implemented as two (or more) parts, mandatory and optional.

4 Relationships among Different Types of On-Board Maintenance

As the above descriptions show, preventive maintenance, perfective maintenance, and corrective maintenance are indeed correlated. In particular,

- Perfective maintenance necessitates corrective

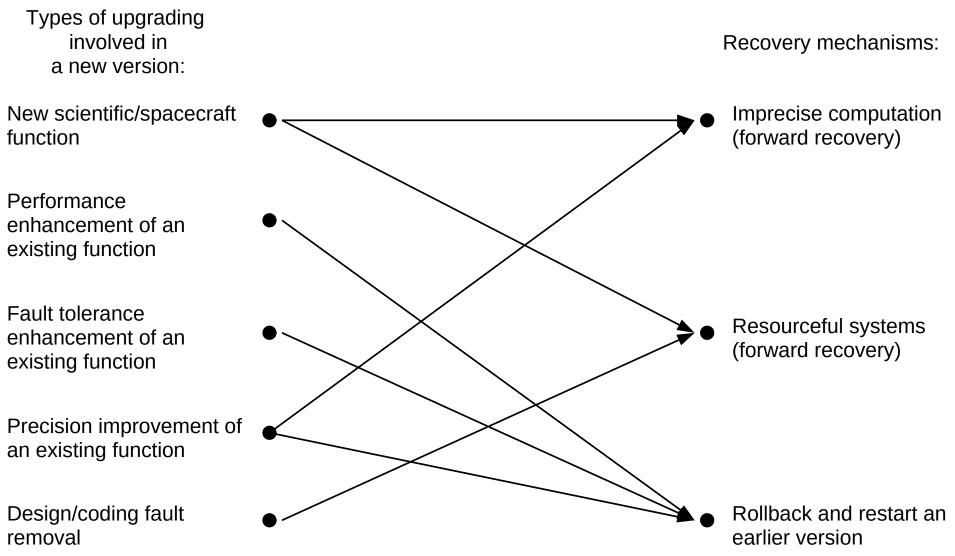
Types of upgrading involved in a new version:
New scientific/spacecraft function
Performance enhancement of an existing function
Fault tolerance enhancement of an existing function
Precision improvement of an existing function
Design/coding fault removal
Recovery mechanisms:
Imprecise computation (forward recovery)
Resourceful systems (forward recovery)
Rollback and restart an earlier version

Figure 4: Upgrading-Adaptive Recovery Rule

maintenance, due to 1) possible inconsistencies between the old and new versions, and 2) the design faults introduced by an added or modified spacecraft/science function.

- Preventive maintenance and corrective maintenance tend to complement each other. In general, corrective maintenance has a higher cost than scheduled preventive maintenance with respect to performance overhead (due to possible rollback, restart, or task re-scheduling). Preventive maintenance eliminates or minimizes the error conditions (in a limited range) that may eventually trigger corrective maintenance or system failure. On the other hand, corrective maintenance covers a wider range of errors and mitigates their effects, including the value errors caused by design faults.
- While preventive maintenance and corrective maintenance provide to a system short-term solutions by mitigating error conditions caused by design faults or other inadequacies in a spaceborne system, perfective maintenance is aimed at the long-term solutions for mission reliability.

Accordingly, the methodologies for realizing the three types of on-board maintenance need to coordinate, in order to mutually enhance effectiveness. Specifically, each type of maintenance shall be made adaptive to the system conditions that are revealed by other types of maintenance. For example, the information from the on-board system logs attained through preventive maintenance and corrective maintenance can be used for designing and scheduling a perfective maintenance for dependability upgrading. Another example is that, after the addition of a computation intensive software function (perfective maintenance), rejuvenation frequency needs to be increased in order to avoid system failures due to design or coding faults in memory allocation. On the other hand, the rejuvenation frequency may be reduced back to "normal" after a sufficiently long period of time during which no such error conditions are observed (through memory-usage log "look-up").

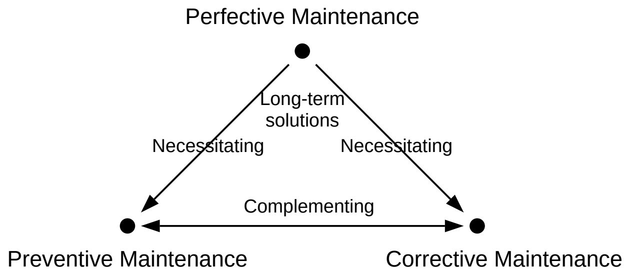

Figure 5: Relationships among Different Types of On-Board Maintenance

An essential step in methodology coordination is to qualitatively predict the interactions among different types of maintenance and to quantitatively evaluate their collective benefits. Of particular interest is to achieve unified measures of system attributes, such as performability [16, 17].

5 Summary

We have described the concept of on-board maintenance. Due to high-cost spaceborne system operation (in terms of power, weight, flight-time limitations), system unavailability resulting from upgrading-related defects or maintenance activities must be minimized, which implies that effective and efficient error detection and recovery mechanisms are crucial. Accordingly, we need to adapt and integrate the state-of-the-art techniques across the areas of fault tolerance, real-time systems and distributed computing, in order to assure reliability and performance gain from on-board maintenance. Currently, we are conducting the corresponding research. As on-board maintenance is becoming a means of dependability enhancement for long-life systems, it is anticipated that the X2000 missions will utilize this new concept and provide to us the effectiveness benchmarks.

References

[1] L. Alkalai and A. T. Tai, "Long-life deep-space applications," *IEEE Computer*, vol. 31, pp. 37–38, Apr. 1998.

[2] L. Alkalai, "NASA Center for Integrated Space Microsystems," in *Proceedings of Advanced Deep Space System Development Program Workshop on Advanced Spacecraft Technologies*, (Pasadena, CA), June 1997.

[3] L. Alkalai and M. Underwood, "Micro-electronics systems IPDT technology roadmap," Technical Report D-13276, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, Dec. 1995.

[4] L. Alkalai, J. Klein, and M. Underwood, "The New Millennium Program microelectronics systems, advanced technology development," in *Proceedings of the 34th Aerospace Science Meeting and Exhibit*, (Reno, Nevada), Jan. 1996.

[5] S. N. Chau, "X2000 avionics system conceptual design document," JPL Technical Report, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 1997.

[6] A. T. Tai, S. N. Chau, L. Alkalai, and H. Hecht, "On-board preventive maintenance: Analysis of effectiveness and optimal duty period," in *Proceedings of the 3rd International Workshop on Object-Oriented Real-time Dependable Systems (WORDS'97)*, (Newport Beach, CA), pp. 40–47, Feb. 1997.

[7] J.-C. Laprie, editor, *Dependability: Basic Concepts and Terminology*, vol. 5 of *Dependable Computing and Fault-Tolerant Systems*. Wien, New York: Springer-Verlag, 1992.

[8] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software rejuvenation: Analysis, module and applications," in *Digest of the 25th Annual International Symposium on Fault-Tolerant Computing*, (Pasadena, CA), pp. 381–390, June 1995.

[9] S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi, "On the analysis of software rejuvenation polices," in *Proc. 12th Annual Conference on Computer Assurance (COMPASS'97)*, (Gaithersberg, MD), June 1997.

[10] S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi, "Analysis of preventive maintenance in transaction based software systems," *IEEE Trans. Computers*, vol. 47, pp. 96–107, Jan. 1998.

[11] P. S. Ho and T. Kwok, "Electromigration in metals," *Reports on Progress in Physics*, vol. 52, pp. 301–348, Jan. 1989.

[12] K.-N. Tu, J. W. Mayer, and L. C. Feldman, *Electronic Thin Film Science for Eletrical Engineers and Materials Scientists.* Maxwell Macmillan International, 1992.

[13] A. T. Tai, L. Alkalai, and S. N. Chau, "On-board preventive maintenance for long-life deep-space missions: A model-based evaluation," in *Proceedings of the 3rd IEEE International Computer Performance and Dependability Symposium*, (Durham, NC), pp. 196–205, Sept. 1998.

[14] A. Avižienis, "Towards systematic design of fault-tolerant systems," *IEEE Computer*, vol. 30, pp. 51–58, Apr. 1997.

[15] J. W. S. Liu, K.-J. Lin, W.-K. Shih, A. C. Yu, J.-Y. Chung, and W. Zhao, "Algorithms for scheduling imprecise computation," *IEEE Computer*, vol. 24, pp. 58–68, May 1991.

[16] J. F. Meyer, "On evaluating the performability of degradable computing systems," *IEEE Trans. Computers*, vol. C-29, pp. 720–731, Aug. 1980.

[17] A. T. Tai, J. F. Meyer, and A. Avižienis, *Software Performability: From Concepts to Applications.* Boston, MA: Kluwer Academic Publishers, 1996.