Proceedings of the Sixth National Conference on Ada ® Technology

March 14-18, 1988

Sponsored by
United States Army
United States Navy
United States Air Force
United States Marine Corps
Ada Joint Program Office

Co-Hosted By
Norfolk State University and University of Maryland

DESIGN FOR FAULT TOLERANCE AND PERFORMANCE IN A DoD-STD-2167 Ada PROJECT

WALTER SOBKIW and THOMAS L.C. CHEN
E-Systems, Inc., ECI Division at St. Petersburg, Florida

ABSTRACT

As computer hardware decreases in cost, it becomes increasingly available to software developers and computer users. With this decrease, computer-based system tools are finding new automation applications, which tend to replace manual tasks. Most of these automation tasks require timely arrival and accuracy of their product or output.

This has resulted in the development of new requirements related to data integrity and system availability. The solution to preserving data integrity and providing for high availability has been to develop fault tolerant computer-based systems. This paper defines a fault tolerant system design methodology within the framework of DoD-STD-2167 and the constraints of implementation in Ada.

INTRODUCTION

The formal system methodologies utilized in the development of many of today's medium-to-large systems tend to only address the functional system requirements. They are usually top-down oriented methodologies which rely on structured systems analysis as developed by Yourdon and or Hierarchical Input Processing Output (HIPO) analysis as pioneered by IBM.

The problem with these methodologies is that they tend to ignore other aspects of the system equally important to accomplishing the system mission. Specifically performance and fault tolerance in the system are not addressed in a methodical manner. In addition, the impact of various issues related to fault tolerance are not related to the functional analysis or the performance analysis of the system.

In the structured system analysis methodology, performance is considered to be a minor issue. The advocates generally assume 10% of the units need to be redesigned in any project to support any unexpected computer performance deficiencies. DoD-STD-2167 defines a system development methodology which relates system development products to major program milestones, however, the standard tends to only focus on the functional definition of the system with little emphasis on performance analysis and no emphasis on fault tolerant analysis.

This point of view has been disputed by many practitioners and system users. There are several articles documenting practitioners and system users point of view. These articles suggest that major effort must be dedicated to fault tolerance and computer performance in the early phases of a design project, and that this analysis be refined as the system design baseline matures.

To a very large extent, the maturation of the design baseline is heavily dependent upon the fault tolerant analysis and computer performance analysis of the system.

In the past, design for fault tolerance and performance of computer-based systems has been successfully achieved without a widely accepted design methodology. Examples of such systems include Nuclear Power Plants, the U.S. Air Traffic Control System.

Experience on an Ada fault tolerant communications system at E-Systems as well as experience in the design of the new U.S. Air Traffic Control System suggests that the design for fault tolerance and performance can be systematically accomplished by a series of analysis. These analysis are aimed at a certain class of questions to produce a set of documented design alternatives as the functional analysis of the system is performed.

These documented design alternatives can be eliminated or refined by the restrictions provided by the computer performance analysis, fault tolerant analysis, design restrictions of the hardware and operating system characteristics, and the functional analysis of the system. These studies and trade-offs must occur in a timely manner in conjunction with the functional design of the system to form the overall system architecture. To some extent the level of detail at each phase is driven by the 2167 standard.

The products associated with the design of a fault tolerant system include an availability model, a computer performance analysis, and a failure analysis. These products each take on a different form and address specific issues at the 2167 major milestones. Many of the fault tolerant key issues will only surface after a methodical detailed analysis of the system has occurred. The specific implementation of Ada is one of the key issues. The required solutions for many projects is several orders of magnitude larger than the solution depicted by the implementation independent function analysis of the system.

With the introduction of Ada, certain constraints are introduced into the fault tolerant design of the system. These constraints are generic in nature, related to the language and implementation-specific related to each vendors particular interpretation of the Ada specification. For example, the use of rendezvous as opposed to semaphore is a constraint on the design generic to the language definition, while the particular design characteristics of tasking for each vendor type is dependent upon that vendor's Ada implementation

PRESENT SYSTEM DESIGN PROCESS OVERVIEW

The present system design process is an i~~n~~tera~~c~~tive process in which certain key issues of the system design are surfaced as the level of detail of the design evolves. The design at the system level can be summarized as follows:

- Requirements Analysis
- Functional Analysis
- Functional Allocation

It is at this time, during system architecture development, that the fault tolerant strategy must be decided. For example, will the fault tolerant strategy be based on hardware, software, or both? Will the hardware and software in the system be built from the ground up or will the system be built using a combination of off-the-shelf hardware and software and newly developed hardware and software?

For example, Stratus Computer Systems [3], [7] fault tolerant strategy is based on hardware in which all the hardware is replicated in a dual-dual configuration. This fault tolerant strategy may make sense if the driving factor is fault tolerance in hardware as opposed to software. If there are issues associated with reconfiguration and recovery time, then August systems [6], [8] TMR system may be of interest since its reconfiguration and recovery is inherent in its normal mode of voting operations. If fault tolerant software is a primary consideration, then perhaps Tolerant Systems [3] or Tandem Computers [3], [5] with their transaction processing facilities may be more appropriate to the application.

These two vendors also may be attractive if the fault tolerant software may use innovative software error detection mechanisms such as multiple designs processing the same data and comparing the outputs. [10]

If there is a tightly coupled application that requires extensive amounts of computer resources, then perhaps a fault tolerant system must be built from the ground up using IBM mainframe like computers. [1], [4], [8], [9], [11]

The IBM mainframe computers offer extensive hardware error checking capabilities with the CPU logic and multiple instruction engines that can be tightly coupled to shared memory.

As the system design process progresses to lower levels of detail, certain issues related to the use of Ada must be resolved. The first question is, will Ada be appropriate for the application in question? Additional questions would be: Will the entire system be Ada-based or will there be some subsystems implemented in C, for example? Will the same Ada-based computer subsystem share another environment such as Unix and will the Unix environment support another language such as Fortran or assembly? Fault tolerant implementation requires control structures. Will these structures be developed by the applications programmers in Ada or will co-existing operating system services provide these control structures?

These issues are surfaced during the course of designing fault tolerant Ada-based systems. However, the framework of 2167 makes it difficult to document the analysis and results that lead to final requirements definitions that resolve these issues in each particular vendor design.

FAULT TOLERANT COMPUTER DESIGN METHODOLOGY

Figure 1 identifies the major milestones in the 2167 system development methodology. Normally when the standard is reviewed and the major milestones identified, the design and management staff view the design from the main mission function which is to define, develop, and design the automation applications functions. For example, if an air traffic control system is being developed, then the primary focus is identifying what the system is suppose to do (i.e., define the system functions) and how the system will support those mission function (using micros, mainframes, array processors, etc.)

However the analysis of the system in terms of availability and fault tolerances becomes of secondary concern. More important is that the functions associated with a fault tolerant mechanism are ignored. Figure 1 also shows the major milestones in a 2167 program and the kind of activities associated with each major milestone. Missing are the products which show the fault tolerant design baseline at each major milestone and how the design effort arrived at the fault tolerant baseline design.

Figure 2 has identified the DoD-STD-2167 major milestones and mapped a family of recommended activities and products to support the design of a fault tolerant system. These products are summarized as follows:

- Fault Tolerant Design Program Plan
- Generic Fault Tolerant Analysis
- Availability Model
- Static and Dynamic Computer Performance Models
- System, Hardware, Software, and Operational Failure Analysis

	SRR	SDR	SSR	PDR	CDR	IMPLEMENTATION
PRODUCTS	- PRELIMINARY SYSTEM SEGMENT SPECIFICATION	- SYSTEM SEGMENT SPECIFICATION - PRELIMINARY SRS AND IRS	- SRS AND IRS - PRELIMINARY TLDD	- TLDD - PRELIMINARY SDDD	- SDDD - PRELIMINARY IMPLEMENTATION	- RELEASED VERSION
ACTIVITIES	- REVIEW AND MODIFY CUSTOMER A-LEVEL SPECIFICATION - SUPPORT THE DESIGN CONCEPT WITH ANALYSIS	- IDENTIFY SYSTEM FUNCTIONS AND INTERFACES - ALLOCATE SYSTEM FUNCTIONS TO GENERIC HARDWARE SUBSYSTEMS - SUPPORT THE DESIGN CONCEPT WITH ANALYSIS	- IDENTIFY CSCI - ALLOCATE SYSTEM FUNCTIONS TO CSCI - ALLOCATE CSCI TO GENERIC HARDWARE - SELECT LANGUAGE AND SYSTEM CONTROL ENVIRONMENT - SUPPORT THE DESIGN CONCEPT WITH ANALYSIS	- IDENTIFY ALL CSCI AND TLCSC - IDENTIFY ALL COTS HARDWARE AND SOFTWARE - ALLOCATE CSC TO HWCI, MAJOR COMPONENTS AND SUBSYSTEMS	- IDENTIFY ALL LLCSC AND UNITS - ALLOCATE LLCSC TO HWCI, MAJOR COMPONENTS AND SUBSYSTEMS - ALLOCATE UNITS TO ADA PACKAGES, PROCEDURES, AND TASKS - SUPPORT THE DESIGN CONCEPT WITH ANALYSIS	**CODE** - CODE - DATABASE POPULATE **DT&E** - TEST CONCEPT - TEST DEFINITION - TEST PROCEDURES - TEST RESULTS

Figure 1. DoD-STD-2167 Major Milestones and Expected Products

SRR – Fault Tolerant Design Program Plan

The System Requirement Review (SRR) is primarily used to review the customer specification. Normally, the system developer has an initial concept of the system and is able to identify problems associated with the customer specification. These problems include functional, computer processing performance requirements, and problems associated with availability and maintainability requirements. At this time, the primary focus of the effort associated with fault tolerant definition should be an examination of the mission and the availability requirements. Many times the mission can be redefined with less stringent availability requirements and the system can exceed fault tolerant expectations. For example, in the case of the communications systems being developed by E-Systems, the availability requirements were defined with centralized communications queues in mind. Some of these systems can be made more fault tolerant with a distributed queue however, the customer specifications would have made those designs non-compliant with the overall system specifications. By SRR, the program plans should be developed and available to all personnel. This includes a stand-alone

	SRR	SDR	SSR	PDR	CDR	IMPLEMENTATION
PRODUCTS	- MODIFIED A-LEVEL SPECIFICATION REQUIREMENTS - FAULT TOLERANT DESIGN PROGRAM PLAN	- AVAILABILITY MODEL RESULTS - STATIC COMPUTER MODEL RESULTS - GENERIC FAULT TOLERANT ANALYSIS (COOK BOOK)	- REFINED AVAILABILITY MODEL RESULTS - REFINED STATIC COMPUTER MODEL RESULTS - FAILURE ANALYSIS: SEGMENT SPEC AND SRS REQUIREMENTS	- REFINED MODEL RESULTS - DYNAMIC MODEL RESULTS - REFINED FAILURE ANALYSIS: SRS AND TLDD REQUIREMENTS	- REFINED MODEL RESULTS - REFINED FAILURE ANALYSIS: TLDD AND SDDD REQUIREMENTS	- FAILURE ANALYSIS - SOFTWARE AVAILABILITY GROWTH PROGRAM PRODUCTS
ACTIVITIES	- REVIEW CUSTOMER A-LEVEL SPECIFICATION RELIABILITY AVAILABILITY REQUIREMENTS	- INITIATE AVAILABILITY MODEL - INITIATE STATIC COMPUTER PERFORMANCE MODEL - IDENTIFY GENERIC FAULTS AND RESULTING ERRORS - IDENTIFY GENERIC FAULT TOLERANT FEATURES	- REFINE AVAILABILITY MODEL - REFINE STATIC MODEL - IDENTIFY SYSTEM, SUBSYSTEM, AND SOFTWARE FUNCTION FAILURE MODES	- REFINE MODELS - IDENTIFY CRITICAL CSCI, TLCSC, AND HWCI. IDENTIFY FAILURE MODES - IDENTIFY CRITICAL OPERATIONS AND FAILURE MODES - IDENTIFY GENERIC ADA CONSTRAINTS AND SOLUTIONS	- REFINE MODELS - IDENTIFY CRITICAL CSCI, TLCSC, LLCSC, UNITS, AND HWCI. IDENTIFY FAILURE MODES - REFINE OPERATIONS FAILURE ANALYSIS - IDENTIFY ADA IMPLEMENTATION SPECIFIC CONSTRAINTS AND SOLUTIONS	PACKAGE - FAULT TOLERANT PROGRAM PLAN - GENERIC FAULT TOLERANT ANALYSIS - SYSTEM FAILURE ANALYSIS - SOFTWARE FAILURE ANALYSIS - HARDWARE FAILURE ANALYSIS - OPERATIONS FAILURE ANALYSIS

Figure 2. DoD-STD-2167 Major Milestones and Recommended Fault Tolerance Analysis Products

plan to support fault tolerant design. The Fault Tolerant Design Program Plan identifies the inputs to the effort, the product outputs, and the relation of the fault tolerant products to all other products on the program.

SDR – Generic Fault Tolerant Analysis, Preliminary Availability, and Static Computer Performance Models

By Software Design Review (SDR) there should be a generic architecture solution. This solution identifies all system functions, allocates those system functions to generic data processing hardware, identifies the internal and external interfaces and provides supporting data to satisfy the system performance requirements. At this time, there also should be:
(1) supporting analysis in the tradeoffs that show the generic architecture solution; (2) the performance analysis identifying the computer processing performance at the function level; and (3) an availability model.

This is a very critical time period for the definition of the fault tolerant system, since effectively the basic approach to fault tolerance is chosen. This definition should include, for example, if the system is to be characterized with large numbers of hardware error checkers capable of detecting transient errors or is the system characterized with software capable of detecting errors in processing induced by the hardware or software. This time period has effectively identified the generic approaches to implementing a fault tolerant solution, developed computer performance and availability models, and traded off the various solutions in overall program goals using the inputs from the various models on each fault tolerant architecture approach.

During this time, the program should examine the state-of-the-art in fault tolerance. This should be in the form of a Generic Fault Tolerant Analysis document that identifies various fault tolerant features and their capabilities in terms of dealing with various hardware and software faults and errors. This document also establishes the fault tolerant definitions that will be used throughout the program development effort. These definitions will include various generic ways in which computer-based systems have failed. In addition, if there is some prior generation of the computer-based system in operation, this document identifies some of the more unusual problems encountered in that previous generation computer-based system.

This document needs to be detailed and yet capable of being reviewed and understood in a very short time period. This document effectively forms a 'cookbook' of generic fault tolerant approaches available to the system designers. Just as an A-Spec is developed to identify the mission of the system, this document is developed to identify various fault tolerant functions and their missions.

For example, a Cyclic Redundancy Check (CRC) attached by the software and maintained during all data transport processing can protect mission data to a very high degree from a host of various system faults. That knowledge may not be readily known or understood by all the system designers (nor should it be). In addition, each designer will have an opinion about the effectiveness of each fault tolerant function.

This document will, for example, provide the official program position on the effectiveness of range checking in software versus the use of hardware error checkers in the CPU hardware, even if the program position is based on qualitative analysis instead of quantitative analysis. The issue is to capture the analysis and make the findings available to all designers during system design.

SSR – System Failure Analysis, Refined Models

The SSR is primarily focused on software design. Effectively, the generic architecture defined for SDR has evolved to a more application specific system definition. The functions previously identified are now allocated to Computer Software Configuration Items (CSCI's). Those CSCI's are allocated to the generic hardware architecture configuration. At this time, the language should have been selected along with the system control environment. Will the system be implemented in Ada using a real-time Unix executive, or will the system be partially implemented in Ada and C with a custom designed real-time control executive? Certain specific implementation issues associated with these software design selections need to be identified and surfaced. In some cases, new functions need to be defined to support the top-level software architecture selection issues. At this time, the fault tolerant design of the system needs to continue refining the computer performance and availability models.

In addition, a formal failure analysis needs to be initiated. That failure analysis needs to begin identifying various failure modes associated with all system mission functions, defining the potential harm of each failure mode (some failure modes may be harmless), and modifying the system baseline from SDR to minimize the potential harm. The output of this failure analysis should be requirements that are incorporated into the system SRS's and the System Segment Specifications.

This failure analysis should be partitioned into system, hardware, software, and operations. The failure analysis should use the design products developed to date and primarily focus in on the SDR baseline design. The most important products are the identification of subsystems, major components, software functions in the form of data flow diagrams, and any single thread diagrams.

The failure analysis should begin by effectively performing a Black box analysis on the automation system where subsystems are removed from operational service because of a failure. Failed subsystems that result in damage to on-line operations and impact the availability requirements are the most critical subsystems. When a subsystem is removed from service, the impact of that event should be documented. Those impacts include loss of system data and mission functions. If critical data or critical mission functions are lost, then various solutions should be identified for preventing the loss of that data or system function service. A starting point for all fault tolerant solutions should be with the generic fault tolerant features identified in the fault tolerant analysis effort used to support the SDR baseline. Those generic solutions should then be modified to support the specific characteristics of the baseline.

Those approaches should be documented and a tradeoff in terms of the effectiveness of each solution and its impact on the system architecture identified. Many times the various fault tolerant solutions have large impacts on the computer performance characteristics of the system, and so the timing and sizing analysis needs to be closely coupled to this tradeoff analysis.

The primary focus at this time should be the system failure analysis, however, the hardware, software, and operational failure analysis can also begin at this time. The other failure analysis efforts will be refined during PDR.

PDR – Full Failure Analysis, Refined Availability Model(s) and Dynamic Computer Performance Model(s)

The Preliminary Design Review (PDR) consists of an architecture where all the interfaces are defined and allocated to a system configuration baseline, all the CSCI's, and Top-Level Computer Software Components (TLCSC's) are identified and their requirements are documented in the Top-Level Design Document. At this time, the computer performance model should have evolved to represent the design in terms of the TLCSC's.

In addition, a preliminary dynamic model representing various queue relationships and context switching baselines should be providing the designers with further definition of the system performance characteristics. The availability model should be tracking the evolving baseline. The architecture solution should be defined in terms of a vendor implementation if off-the-shelf hardware and or software will be used in the system.

The fault tolerant analysis should now switch to a failure analysis of the each individual TLCSC and Hardware Configuration Item (HWCI) in the system. The TLCSC's should be characterized in terms of importance in the system and the various failure modes previously identified at SDR should be allocated to the TLCSC's identified. In addition, each HWCI should be characterized in terms of importance in the system, and the various subsystem failure modes previously identified at SDR should be allocated to the identified critical HWCI's.

It is at PDR that the architecture solution has transitioned from a generic solution to a non-generic solution. The Ada language has been selected along with any Commercial Off-the-Shelf (COTS) hardware and software. In addition, the design of new hardware and software has begun to be constrained by various implementation issues.

For example, if a high speed communication bus is being developed the access mechanism should be defined. Is the bus Carrier Sense Multiple Access (CSMA) or Token controlled? Is the physical plant fiber optic or copper based? The same non-generic issues related to fault tolerance also should be surfacing at this time. Which TLCSC's and HWCI's are critical? For the critical TLCSC's and HWCI's, what are the error detection mechanisms? For the critical TLCSC's and HWCI's, what are the recovery mechanisms? How long does it take to recover and will the availability be satisfied? These are all very detailed questions that cannot be reasonably answered without a non-generic architecture definition.

It is at PDR that the generic constraints of Ada begin to surface. Fault tolerance is accomplished with error detection, replication of the same design or alternate designs, and routing via active healthy paths. In the non-generic architecture, these functional requirements translate to control structures.

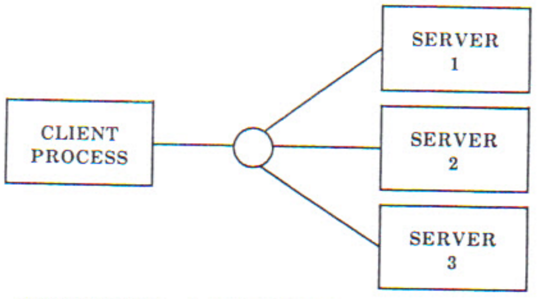
For example, to protect data from damage resulting from memory storage failure data is replicated. This replication can occur in hardware or software. However, after examining the market place there are no vendors that offer full data replication implemented in hardware. This control structure must either be implemented in the applications code or the operating system services.

In addition, to maintain consistency before, during, and after a failure, the mechanism grows into a sophisticated sequence or control structures filled with many potential design errors in and of itself. In addition, once an error is detected by either hardware or software, recovery must proceed. This recovery also must be in a consistent manner to ensure that system data is not compromised. These are all elaborate control structures that must be implemented in Ada by the applications programmer from the ground up.

This issue is further exacerbated by the context switch time of many Ada implementations. In order to implement many of these control structures in Ada context switching needs to occur. This context switch time can significantly reduce the time left to support application programming functions. This issue is particularly acute in real-time fault tolerant system design.

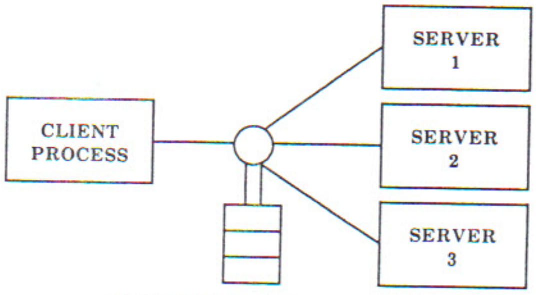
Some vendors such as Tolerant systems have developed control structures and made them available to the applications programmer. In Tolerant's case, these control structures reside in an augmented Unix look alike operating system. The Ada 'application' code executes with the augmented Unix look alike operating system.

It is at this time that the issues related to interprocess communications and process synchronization surfaces. There are two basic process synchronization schemes as depicted in Figure 3.



RENDEZVOUS – A BASIC MESSAGE PASSING PROCESS SYNCHRONIZATION

- A CLIENT MUST WAIT FOR SERVER, SERVER MUST WAIT FOR CLIENT.
- IN A UNI-PROCESSOR SYSTEM, A CONTEXT SWITCH IS ALWAYS INVOLVED.



SEMAPHORE – A BASIC SHARED PROCESS SYNCHRONIZATION

- PRODUCER PLACES DATA IN QUEUES IN SHARED STORAGE; CONSUMER REMOVES DATA FROM QUEUES.
- PRODUCER NEVER WAITS FOR CONSUMER AND CONSUMER NEVER WAITS FOR PRODUCER.
- CONTEXT SWITCH OCCURS ONLY WHEN THERE IS A COLLISION ON THE SEMAPHORE WHICH IS VERY INFREQUENT IN THE REAL WORLD.

Figure 3. Rendezvous vs True Semaphore

The first one is based on a semaphore and requires shared memory. When this scheme is selected the producer process deposits its products in the shared memory and the consumer process takes the products off the shared memory. The semaphore is used to arbitrate the access of the shared storage. A process context switch is only required when there is a hit on the semaphore. This is an old efficient scheme which requires the support of shared memory. The second scheme is message passing. Two processes are synchronized by expressly passing messages to each other. When both processes share the same hardware processor, a context switch is always required in synchronization. When each process is supported by a different hardware processor, one of the processes will be idle waiting for the other process unless the duty cycle of both processes are perfectly matched. Tasking in Ada is synchronized by rendezvous, which is a message passing scheme.

In the semaphore scheme, the application context and the queues need to be secured for fault tolerance. In the message passing scheme, the processes (task table, stack, etc.), as well as the application context, has to be secured for fault tolerance.

A project can chose to use Ada tasking or an operating system process in place of Ada tasking. The choices of Ada will also effect the selection of the hardware. No Ada compiler in existence today can allocate Ada tasks to a different hardware processor nor is there any support for rendezvous across hardware processors. The nature of the application may favor one scheme or the other. The hardware architecture should match the selected scheme. The architecture design must weigh all these interdependent matters to form the lowest cost solution that is technically practical. Many designers would argue that this is an implementation issue, however, these two choices can severely impact the architecture to the point of invalidating the architecture concept.

CDR – Refined Products ready for delivery to Software Availability Growth Organization

By CDR, the requirements, analysis, and design documentation should be complete. This milestone effectively gives the go-ahead for the contractor to begin implementing all the hardware and software. It is at this time that the Ada packaging and procedure concepts should be allocated to the TLCSC's, LLCSC's, and units.

At this time, the failure analysis has focused on the units, HWCI's, and resolved all issues related to the system databases. The issues related to Ada packages and procedures also have been resolved with a clear indication of how each package and procedure will be supported with fault tolerant features. The error detection, reconfiguration, and recovery mechanisms should be fully documented as requirements in the specifications and those requirements should be justified in the availability model and analysis, dynamic model and analysis, and the failure analysis.

The failure analysis should have identified all the design constraints associated with developing a real-time fault tolerant Ada system using the selected hardware and accompanying software. This analysis will then be submitted to the organization tasked with the software availability growth program. This software availability growth program should be fully established and ready to begin its activities with the test and integration phase during implementation.

It is at this time that the vendor implementation of Ada will begin to constrain the design of the system. For example, the Ada task is not constrained in DoD-STD-1815 and there are a number of different vendor implementations of this Ada task some of which do not support true parallel or concurrent processing. This issue is particularly acute in a communications system where concurrent I/O is to be supported. There are also issues related to the Ada load image and the management of the heap. In both cases, memory is consumed such that embedded real-time fault tolerant systems can be severely constrained. These issues need to be surfaced and resolved by CDR.

CONCLUSIONS

In conclusion, the design of a fault tolerant system is similar to the design of the main functional application of that real-time system. The fault tolerance analysis needs to begin with a plan. That plan should not only contain the approach and methodology to supporting the fault tolerant design, but it should also contain a heavy emphasis on the technical aspects of fault tolerance. Once this plan and cook book are established, then a failure analysis of the system needs to occur at each level of the system design. This begins with the subsystems and top-level functions and concludes at the Ada packages or procedures and the HWCI's. The entire failure analysis should then be packaged as one product and submitted to the software availability growth program organization.

During the course of the design, many control structures used to support fault tolerance will need to be developed. Some of these control structures have been implemented in external operating system environments. Other structures will need to be implemented in Ada and will be limited by context switching time. In some cases, some of the fault tolerant control structures may need to be developed in assembly language to preclude the negative impacts of context switching time.

As with all design methodologies, the intent is to provide a vehicle for identifying and controlling program risk. Risk in this case is related to the successful design of a fault tolerant system. With the identification of these products and related activities schedule, cost, resources, and expertise can be identified and planned to support a successful fault tolerant design. More importantly, the progress of the fault tolerant design activity can be effectively tracked during the entire design process.

REFERENCES

1. IBM Journal of Research and Development, IBM 3081 System Development Technology Vol 26, Number 1, Jan 1982.

2. Computer System Isolates Faults - The Tolerant Systems Eternity Series, Computer Design, Nov 1983.

3. Fault Tolerant Systems in Commercial Applications, Omri Serlin. IEEE Transactions on Computers, Aug 1984, pp 18-30.

4. Fault Tolerant Computing - Concepts and Examples, David A. Rennels,Vol C-33, No. 12, pp 1116-1129. IEEE Transactions on Computers, Aug 1984.

5. Fault Tolerant Architectures Douglas Eidsmore, Digital Design, pp 70-82, Aug 1983.

6. Fault Tolerant Computers Ensure Reliable Industrial Controls Electronic Design, 25 Jun 1981.

7. Making Processing Fail-Safe Robert Fredburg, pp 255-264. Mini-Micro Systems, May 1982.

8. Fault Tolerant Computer Study, Jet Propulsion Lab, JPL Pub 80-73, Contract NAS 7-100, pp 2-1 to 2-51.

9. Survey of Fault Tolerant Computer Security andComputer Safety SRI International, NTIS RADC.TR-86-164 I-1 to I-26, IV-1 to IV-56.

10. An Empirical Study of Software Error Detection Using Self-Checks. Fault Tolerant Computing Symposium July 1987. Sung D Cha, John C. Knight, Nancy G. Levenson, and Timothy J. Shimeall, pp 156-162.

11. Fault Tolerance Principals and Practice, T. Anderson and P.A. Lee Computing Laboratory, University of Newcastle upon Tyne, England. I-1 to III-89.

Biography

Walter Sobkiw is a Senior Principal Engineer with E-Systems, ECI Division. He is directly responsible for failure analysis, simulation testing, and automation systems for military communications systems. He holds a BSEE from Drexel University.

Thomas L.C. Chen is a Member of the Technical Staff in the Software Systems Department, E-Systems, ECI Division. He is the Principle Software Designer of Survivable Communications Systems, has over 20 years experience in the development of communications methodology. He holds an M.E. from Taipei Institute of Technology.