25th FTCS Special Issue

Building Dependable Systems: How to Keep Up with Complexity

Algirdas Avižienis

University of California, Los Angeles, CA, 90095 USA
and
Vytautas Magnus University, Kaunas, Lithuania

ABSTRACT

This paper reviews the origin of the concept of fault tolerance and the evolution of guidelines for the systematic design of fault-tolerant systems. The current formulation of the guidelines, called a design paradigm, is presented. The problem of using off-the-shelf subsystems in a fault-tolerant system is discussed. In conclusion, an analogy of complex fault-tolerant systems and living organisms is suggested as a means to advance the understanding of fault tolerance.

1. Origin of the Fault Tolerance Concept

The concept of fault tolerance originally appeared in technical literature in 1967 as follows [1]:

"We say that a system is fault-tolerant if its programs can be properly executed despite the occurrence of logic faults."

The creation of the fault tolerance concept was the consequence of the convergence of three developments.

First, there were the ongoing efforts by computer designers to increase the reliability of computing by the use of various practical techniques. As soon as the first computers were built and turned on, it became apparent that physical defects and design mistakes could not be avoided by careful design and by the choice of good components alone. For this reason, designers of early computers employed redundant structures to mask failed components, error control codes and duplication or triplication with voting to detect or correct information errors, diagnostic techniques to locate failed components, and automatic switchovers to replace failed subsystems [2, 3, 4, 5].

Second, in parallel with the evolution of the engineering techniques, the general problem of building reliable systems from unreliable components was addressed by some of the founders of computer science, most eminently J. von Neumann in 1952 [6], E.F. Moore with C.E. Shannon in 1956 [7], and others.

Third was the new challenge of building unmanned spacecraft for interplanetary exploration that was assigned by NASA to Caltech's Jet Propulsion Laboratory (JPL) in late 1958. Mission lengths of up to ten years and more were being considered, and on-board computing was a prerequisite for their success. Design of computers that would survive a journey of several years and then deliver their peak performance at a distant planet was an entirely unexplored discipline.

Existing theoretical studies of the long-life problem indicated that large numbers of spare subsystems offered a promise of longevity, given that all spares could be successfully employed in sequence. The JPL problem was to translate the idealized "spare replacement" system model into a flightworthy implementation of a spacecraft guidance and control computer.

A proposal to design such a computer, called "A Self-Testing-And-Repairing System for Spacecraft Guidance and Control," and designated by the acronym "STAR" was presented in October, 1961 [8]. It was supported by JPL and accepted by NASA management, and the research effort continued for more than ten years, culminating with the construction and demonstration of the laboratory model of the JPL-STAR computer [9]. The originality of the concept was recognized by U.S. Patent No. 3,517,671 "Self Testing and Repairing Computer," granted on June 23,

0-8186-7146-7/95 $4.00

1970 to A. Avižienis and assigned to NASA. A flight model of the JPL-STAR was designed for a 10-15 year space mission, but its building was halted when NASA discontinued the Grand Tour mission for which it was intended [10].

The longevity requirement led to the study of all accessible engineering solutions and theoretical investigations of reliability enhancement. The variety of existing theories and techniques motivated the definition of the unifying concept of fault tolerance that merged diverse approaches into a cohesive view of all system survival attributes, and greatly facilitated the design of the JPL-STAR computer. It also was a logical step for JPL, the birthplace of the fault tolerance concept, to be the co-sponsor and to take the initiative to organize the first International Symposium on Fault-Tolerant Computing that was held in Pasadena, CA on March 1-3, 1971.

During the next two decades - the 70's and the 80's - we have seen a continuing growth of the universe of faults that are to be tolerated by fault-tolerant systems. The original concept dealt with transient and permanent logic faults of physical origin. Faults due to human mistakes in design were added when the growing complexity of software and of logic on VLSI chips made the removal of all *design faults* prior to operational use not certain. Experience also led to the addition of *interaction faults,* inadvertently introduced by humans during the operation or maintenance of a computer.

Finally, consequences of malicious actions intended to alter or to stop the service being delivered by a system were recognized as being *deliberate design faults.* This concept establishes a common ground for the unified treatment of security and fault tolerance concerns in system design. The assurance of full compatibility and integration of security and fault tolerance techniques is a major challenge for contemporary designers. An overview of the current universe of potential faults is presented in Figure 1.

In retrospect, it may be said that the concept of fault tolerance has served well during the past quarter of a century in facilitating the appearance of successively more dependable systems for the control and support of various essential functions of contemporary society: computing and communications, transportation, nuclear power, financial transactions, health care delivery, etc.

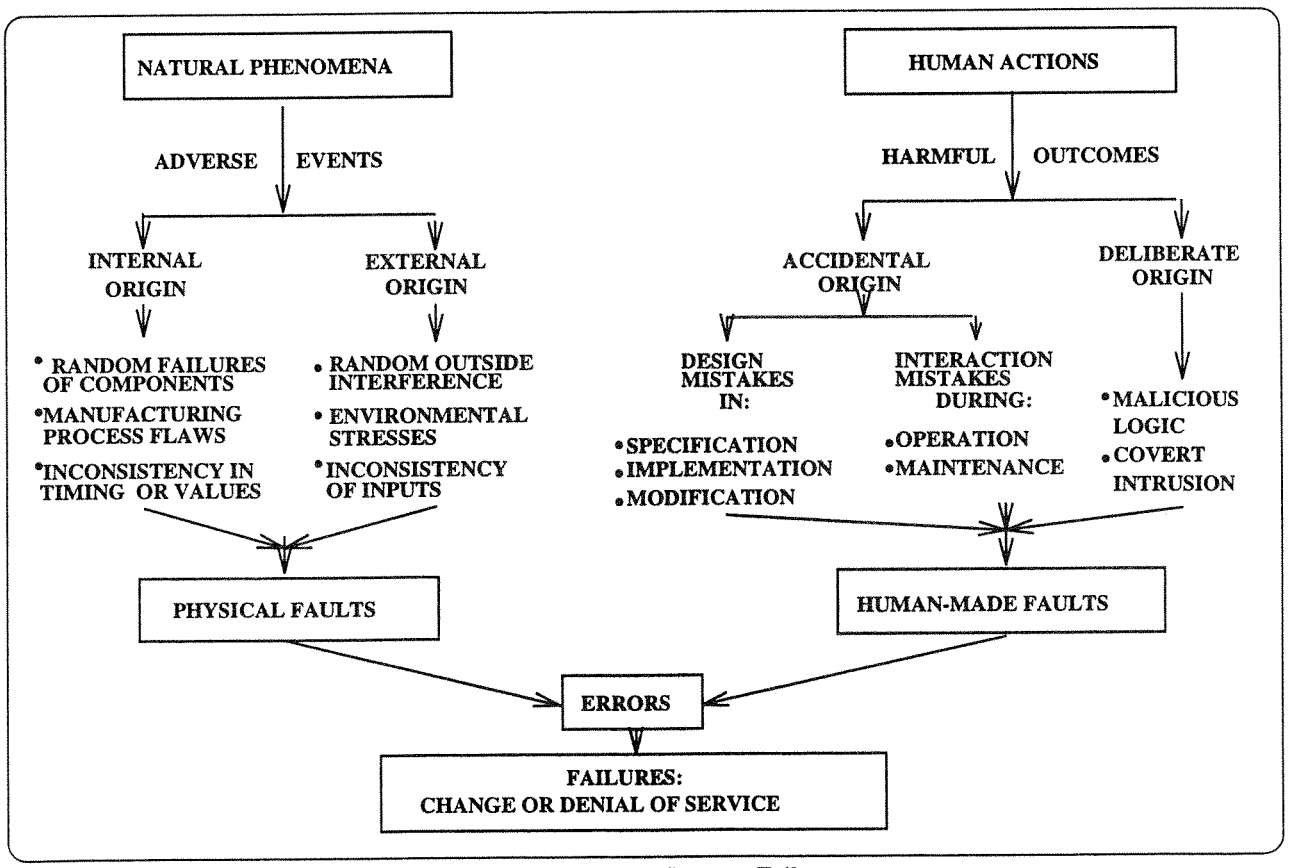

Figure 1: Why Systems Fail

2. Systematic Design of Fault-Tolerant Systems

Thirty years of experience have shown that the building of dependable systems requires the balanced use of both fault avoidance and fault tolerance techniques. An imbalance in either direction leads to an ineffective use of resources and severely limits the attainable dependability. The definition of the concept of fault tolerance initiated the evolution of principles for the systematic design of fault-tolerant systems. The specification and design of the STAR computer at JPL involved much improvisation and experimentation with design alternatives. It became apparent that the lessons learned during this process could serve as the foundation for a more orderly approach that would utilize a set of guidelines for the choice of fault masking, error detection, fault diagnosis, and system recovery techniques.

The first effort to devise such guidelines was presented at the 1967 Fall Joint Computer Conference in the paper "Design of Fault-Tolerant Computers" [1] that is reprinted following this paper. That paper first introduced the term "fault-tolerant computer" and the concept of "fault tolerance" into technical literature. It also presented a classification of faults and outlined the alternate forms of masking, diagnosis, and recovery techniques along with some criteria for choices between "massive" (i.e., masking) and "selective" application of redundancy. The design of the JPL-STAR computer was used to illustrate the application of these criteria in choosing the fault tolerance techniques for a spacecraft computer that had long life and autonomy requirements with strict weight and power constraints. The 47 references covered the most relevant published work to mid-1967.

The earlier book "Failure-Tolerant Computer Design" by W.H. Pierce [11] served as an important reference; however, Pierce's definition of "failure tolerance" corresponded exactly to fault masking in logic circuits, including voting, adaptive, and interwoven logic, redundant relay contact networks, and application of error correcting codes as a masking technique. It is a definitive work on the masking forms of redundancy that were known at that time. However, neither error detection, nor fault diagnosis, nor recovery techniques were included as elements of Pierce's "failure-tolerant" computers.

The 1967 paper was the first of a sequence of publications that formulated an evolving view of how to attain dependable computing by the judicious introduction of fault tolerance during system design. Two different classes of faults - those due to physical causes and those due to human mistakes, oversights, and deliberate actions are considered. This evolving view was presented in a series of papers on guidelines for fault-tolerant system design and implementation, supported by specific discussions of the techniques, scope, and aims of fault tolerance and fault avoidance in hardware, software, communication, and man/machine interfaces. Milestones of this series have been the papers: [12, 13, 14, 15, 16, 17]. Strong motivation for the effort came from the increasing number of successful fault-tolerant systems that offered new design insights and more operational experience.

3. A Design Paradigm

The unifying theme of the above referenced work over the past 27 years has been the evolution of a design paradigm for fault-tolerant systems that guides the designer to consider fault tolerance as a fundamental issue throughout the design process. The word "paradigm" is used here in the dictionary sense of "pattern, example, model" in place of the word "methodology" that implies a study of methods, rather than a set of guidelines with illustrations that is discussed here.

Taken in order of appearance, the papers show a progressive refinement of concepts and an expansion of the scope to include the tolerance of "human made" design and interaction faults. Other recently introduced themes are the balancing of performance and fault tolerance objectives during system partitioning, and the integration of subsystem recovery procedures into a multi-level recovery hierarchy. Strong emphasis has been directed to the application of *design diversity* in a multichannel system in order to attain tolerance of design faults, [14, 17], including the tolerance of deliberate design faults [18].

Fault tolerance has now been recognized as the key prerequisite of dependability for very large systems, such as the FAA's Advanced Automation System for air traffic control [19]. Because of their great functional complexity, such systems pose the most severe challenge yet in the design of fault-tolerant systems. The introduction of fault tolerance into very complex, distributed systems is most likely to succeed if a methodical approach is employed. This approach begins with the initial design concepts and requires the collaboration of performance and fault tolerance architects during the critical tasks of system partitioning, function allocation, and definition of inter-subsystem communication and control. Such a

design approach is presented here as the *design paradigm* for fault-tolerant systems.

The design paradigm is an abstraction and refinement of observed design processes, in which the various steps often overlap. Its objective is to minimize the probability of oversights, mistakes, and inconsistencies in the process of meeting the specified goals of dependable service with respect to defined classes of faults by means of the chosen implementation of fault tolerance. The paradigm is stated for the implementation of a new design. If the goal is the improvement of an existing design, then each step is a reexamination, possibly leading to changes of previously made decisions.

SPECIFICATION

(1) Dependability of Service
(2) Classes of Expected Faults
(3) Evaluation Methods

DESIGN & EVALUATION

(1) System Partitioning:

* Choose Containment Boundaries
* Allocate Redundant Resources
* Decide Design Diversity
* Assign Global F.T. Functions
* Set Subsystem Goals

(2) Subsystem Design:

* Error Detection
* Fault Diagnosis
* Error Recovery
* Fault Removal
* Recovery Validation
* Evaluation

(3) Systemwide Integration:

* Qualitative Evaluation
* Simulations
* Prototyping
* Experimentation
* Quantitative Evaluation

Figure 2: A Design Paradigm

The paradigm partitions the system building process into three activities: *specification*, *design*, and *evaluation*. Design consists of *system partitioning*, *subsystem design*, and *system integration* steps. *Evaluation* takes place during and after each design step. The principal steps of the paradigm are summarized in Figure 2.

3.1 Specification

The specification activity begins with the detailed definition of *system requirements* that describe the services to be delivered in terms of functionality and performance. Dependability-related aspects of service that need to be identified are:

(1) The existence of different *mission phases*, characterized by different environments and conditions of operation during the lifetime ("mission") of the system.

(2) The varying *criticality* of service delivery and system *safety* and *security* goals during different mission phases.

(3) The acceptability of possible different *modes of service* (full, reduced, degraded, emergency, safe shutdown, etc.) during each phase.

(4) *System availability* goals for each mission phase, expressed in terms of the maximum allowed frequency and duration of service mode reductions due to fault occurrence, including time bounds on subsequent recovery actions.

(5) Conditions for the *renewal of resources* (external repair by replacement and remote fault tolerance support) for different mission phases: "on demand" renewal, periodic renewal, no renewal at all.

(6) System *longevity goals*, expressed in terms of reliability and maintainability requirements.

The *second* aspect of the specification activity is the identification of the *classes of faults* expected to occur during each mission phase. Fault classes may differ because of differences in environmental conditions and operating interfaces. There are two parts of this problem:

(1) Internal faults that can arise within the system during each mission phase.

(2) External faults that can introduce errors into the system via the interfaces (I/O links, external fault tolerance support, human/machine interaction) during each mission phase.

The *third* aspect of specification is to choose the *evaluation methods* that will be applied to assess the likelihood that the design will meet the specified

dependability goals: availability, reliability, safety, security, maintainability. The independent means of validating the evaluation also need to be agreed upon and specified at this time. Fault and error occurrence scenarios need to be defined for the purpose of evaluation, including two (or more) independent near-coincident faults with overlapping effects, and the existence of latent errors.

3.2 Partitioning

The first stage of design is the definition of system architecture, expressed by the structure of its building blocks and communications links. This process is here called partitioning. Functionality, performance and fault tolerance requirements, available technologies, and past experience all affect the choice of the hardware, software, communication and interface subsystems that comprise the system being designed.

This step of the paradigm requires a close collaboration between the two groups of system architects: those responsible for performance and those responsible for fault tolerance. The partitioning that is accomplished should be a balanced solution that accommodates both sets of requirements (performance and dependable operation) at the same time.

Several fundamental decisions regarding the implementation of fault tolerance need to be made here. First, partitioning defines a set of fault and error containment boundaries. The purpose of the boundaries is to assure the independence of failure of individual subsystems.

Second, the choice is made of the methods to introduce redundant hardware and software resources. Major options are: (1) multiple channel computation: duplex, triplex, etc., (2) spare subsystems, (3) degradation by exclusion of failed subsystems. Combinations of these techniques may be selected as well. The choice of time redundancy (repetition of computations) is also a possibility.

The third major decision is whether *design diversity* of redundant resources is to be implemented in some or all software and/or hardware subsystems. Design diversity is an especially effective means to assure safety in critical applications and to protect the most vital subsystems of complex systems and networks.

The fourth decision is whether detection, diagnosis, and recovery functions (discussed in the next section) are to be (1) localized within individual subsystems; (2) shared by a set of functionally identical subsystems; (3) extended over a group of functionally diverse subsystems; or (4) made a global function of all subsystems. A hierarchical arrangement of two, three, or all four of these approaches offers other solutions of the detection and/or recovery problem.

After the preceding decisions have been made, every subsystem is characterized by its own error and fault containment requirements, as well as the participation requirements in set, group, and/or global detection and recovery functions discussed in the preceding paragraph. At this time the system-level goals for availability, reliability, maintainability, safety, and security (discussed in 3.1 above) can be apportioned among the independent sets of functionally equivalent subsystems that compose the entire system.

3.3 Subsystem Design

At the completion of partitioning each subsystem is characterized by a set of requirements for functionality, performance, and dependability. The dependability requirements include classes of faults to be tolerated, boundaries for error and fault containment, diversity objectives, modes of service, availability in terms of limits on the frequency and duration of service mode reduction and/or complete outage, and the bounds on recovery time. Furthermore there are the goals for subsystem reliability, maintainability, safety and security that are derived from the system-wide goals. Finally, the subsystem may be required to take part in *external* (i.e., group, set, or global) error detection, fault diagnosis, and/or recovery. The dependability goals are supported by diverse or "cloned" replication at subsystem level and by built-in (*internal*) error detection, fault diagnosis (location), and recovery algorithms.

Errors are undesired states that are caused by active faults and may lead to subsystem failure. Error detection methods fall into two categories: (a) *concurrent* with delivery of service; and (b) *preemptive*, requiring that service delivery should be suspended during the application of detection procedures. Error detecting codes are an example of the former, and memory "scrubbing" an example of the latter method.

In addition to detection, it is necessary to *record* the detected erroneous state for analysis that will facilitate fault diagnosis. The main methods of fault diagnosis are: (a) analysis of a detected erroneous state; (b) repetition of previous operation(s); (c)

application of test patterns; (d) use of special circuitry: microdiagnostics, scan-in/scan-out, etc.; (e) use of independent (monitoring) subsystems. A special error detection and fault diagnosis requirement is that it is necessary to detect errors produced by faults in the fault tolerance mechanisms themselves. Furthermore, provisions are needed to check spare subsystems for dormant faults and to detect latent errors in stored information that is not subject to other forms of error detection.

The choice of specific error detection and fault diagnosis methods strongly depends on the function of the subsystem. For example, error control codes are very often used in memory, while duplication and comparison is common for processors. The choice and location of error detection mechanisms must minimize the probability of undetected error propagation across subsystem boundaries. Additional detection and diagnosis techniques may be incorporated to support externally implemented (group, set, global) fault tolerance functions.

Every error signal that originates in a subsystem must invoke an *error recovery* procedure that attempts to restore the system to a valid state without *known* errors. Error recovery can be: (a) *backward,* returning the system to a previous, error-free state; or (b) *forward,* constructing a valid, error-free new state from existing (usually redundant) information.

A persistent fault may prevent successful error recovery. In this case, fault diagnosis identifies a faulty subsystem, and fault removal is performed by: (a) substituting a good spare subsystem; or (b) reconfiguring the system to continue functioning in spite of or without the faulty subsystem. Finally, an *independent validation* that the recovery was successful is a desirable attribute for every subsystem.

Recovery algorithms are usually hierarchical: a fast, local recovery is attempted first. If it does not succeed, a more extensive and time-consuming recovery procedure is invoked, and so on, until either recovery is accomplished, or the system is safely shut down by the last procedure of the recovery hierarchy. The time that is available to complete a recovery strongly depends on the application of the system. For example, a complete repetition that is acceptable in financial transaction handling would be too slow in real-time flight control. In addition to its internal recovery procedures a subsystem may also incorporate features that support external (group, set, global) cooperative recoveries.

The adequacy of the chosen internal error detection, fault diagnosis and recovery methods is assessed by an evaluation that takes place during subsystem design and may lead to modifications of initial choices.

3.4 Systemwide Integration of Fault Tolerance

Large and complex fault-tolerant systems have experienced unanticipated, failure-inducing *improper interactions* of otherwise well-designed fault-tolerant subsystems that have not been perfectly integrated into a distributed, fault-tolerant system. Especially difficult to integrate are the various hierarchical fault diagnosis and recovery sequences that support the localized fault tolerance of subsystems and those that support the fault tolerance of functional services ("threads") that are provided by two or more subsystems.

A second major integration problem is presented by two or more *nearly concurrent fault manifestations.* The large size and distributed nature of new systems lead to the possibility of two or more independent fault manifestations occurring close in time, most likely because of the previously undetected existence of latent errors and dormant faults. This, in turn, will require two or more recovery algorithms to be concurrently active, with the resulting risks of mutual interference, deadlocks, and behavior that is very difficult to anticipate.

A successful integration of several fault-tolerant subsystems requires in-depth analysis as well as extensive experimentation, including *incremental demonstrations* of the capabilities, protection, completeness, and consistency of all fault tolerance functions and their specific implementation not only under single, but also under *multiple overlapping* fault conditions. The growing complexity of fault-tolerant systems in critical applications requires extensive use of the most advanced analytic and experimental evaluation techniques in order to eliminate inadequacies of global (systemwide) detection and recovery algorithms.

4. Evaluation

The evaluation of the adequacy of the chosen fault tolerance techniques is a continuous process during the steps of system partitioning, subsystem design, and system integration. At each step analytic and experimental evaluation is an important design tool that facilitates the choices between alternative fault tolerance techniques and assesses the likelihood of meeting the dependability goals.

Successful completion of the design of a fault-

tolerant system requires a convincing verification of two properties: the *completeness* of the design and its *potential to meet the stated goals* of availability, reliability, maintainability, and safety. The verification therefore must consist of the application of two distinct evaluations: first *qualitative*, then *quantitative*, or *numerical* [20].

The *qualitative evaluation* of fault tolerance attributes verifies that all the necessary defenses against the expected classes of faults have been properly incorporated into the design. A study of the preceding steps of the design paradigm leads to the guidelines for a structured qualitative evaluation. It is a "pass/fail" type of examination that employs an "inverse" of the design paradigm, called the *qualitative evaluation paradigm*, to apply a series of searching questions about the completeness and appropriateness of fault tolerance techniques.

It is essential that the qualitative examination and evaluation should be satisfied *prior* to generating numeral predictions of system reliability and availability. Otherwise, unreasonably optimistic predictions can be obtained because of unjustified simplifications that remain unnoticed. Examples are: (1) insufficient fault assumptions, such as omission of design faults and man/machine interaction faults, disregard of nearly concurrent manifestations of two or more faults, disregard of latent errors and dormant faults, etc.; (2) overestimation of the effectiveness and speed of error detection, fault diagnosis, system state recovery, and reconfiguration methods; (3) implicit assumptions that all fault tolerance functions do not contain design faults themselves, that they are fully protected against physical faults, and that they always interact perfectly on a systemwide basis.

The *numerical evaluation* verifies that the quantitative requirements of availability, maintainability, reliability, and safety can be met by the design that has passed the qualitative evaluation. A design first must be described in terms of a system reliability model. This model is characterized by sets of physical, structural, repair, fault tolerance, and performance parameters for every subsystem, including subsystem communication links.

The most critical and difficult task is the determination of the most likely ranges of coverage and execution time parameters for all fault tolerance functions of every subsystem. These functions are: error detection, fault location and removal, state restoration, reconfiguration, and recovery validation. Their execution must be coordinated on a systemwide basis. For this reason, the coverage and execution time parameters have to be predicted for subsystem interactions, assuming that one or more than one overlapping fault manifestation can occur, or that latent errors may be encountered in the course of an attempted recovery.

In order to be able to predict the impact of faults on system performance, the availability of the complete system must be estimated in terms of two distinct measures:

(1) The *Mean Time between Mode Reductions* that indicates the expected frequency of transitions into modes of operation below full service;

(2) The *Duration of Mode Reduction* that indicates the expected length of stay in the less desirable modes of operation, expressed in various measures (mean value, 99th percentile, worst case, etc.).

These measures are derived using the system reliability model that has been validated by means of the qualitative evaluation paradigm. They are very sensitive to the values of the *coverage* and *execution time* parameters of fault tolerance functions. These parameters need to be obtained by means of a sequence of progressively more specific and precise methods: estimates, analyses, simulations, and incremental experimentation with prototypes of critical elements under fault conditions. Proofs of certain design properties also may prove to be useful here. Finally, it is also necessary to assess the validity of evaluation tools and models that were used in an evaluation.

5. The "Off-the-Shelf" Problem

The preceding sections discuss the design of fault-tolerant systems that are composed of an integrated set of fault-tolerant subsystems. However, practical consideration of cost and development time often lead to the use of pre-existing or "off-the-shelf" subsystems, such as microprocessors, displays, sensors, workstations, operating systems, application software packages, etc., as building blocks of systems that are expected to be highly dependable. The off-the-shelf items usually either have limited fault tolerance or none at all.

An example of this kind of system is a Picture Archiving and Communications System (PACS). This system provides radiology data to a hospital and research community at UCLA, archiving images from X-rays, MRI and other advanced scanning devices. PACS consists of a networked collection of medical imaging devices, high performance servers for image processing and storage, optical disk storage devices,

and numerous workstations for display and analysis of radiological data. In addition it is integrated with other networks that make up the Hospital Information System, and Radiology Information System [21].

The PACS system requires a high degree of availability, since downtime makes crucial records such as images unavailable. Downtime of more than a few seconds is very costly. It also must protect the integrity of its database of patient records, many of which are legally required to be kept for many years. Loss of or unauthorized access to any of these records can have serious medical (and potentially legal) consequences. Currently PACS is not fault-tolerant and costly outages often occur.

The problem of "retrofitting" PACS and similar systems for fault tolerance is very difficult. The common approach is to design a "monitor" software subsystem that attempts to check all subsystems for indications of failure, records abnormal symptoms, and initiates reconfiguration when needed. The weakness of this approach is the lack of protection for the monitor software itself, since it must reside and execute in an off-the-shelf processor.

A more fundamental solution for a large heterogenous distributed system such as PACS is to make it more dependable by implementing a smaller, highly fault-tolerant hardware system that monitors its operation, assures the protection of data integrity, and manages recovery when part of the system fails by switching in spare resources or reconfiguring the system [22]. The Test-and-Repair Processor of the JPL-STAR computer [9] is the first prototype of such a hardware monitor. An effort to devise a fault-tolerant hardware monitor for the UCLA PACS is currently in progress.

Another illustration of the difficulty of the off-the-shelf problem is the AAS system for air traffic control in the U.S. [19] Its system level specification stated that "availability shall be the most important consideration in the design of AAS," and fault tolerance was explicitly required as the means to assure very high availability. The allowed time of service denial is about 3 seconds per year for critical functions (below emergency mode), 32 seconds per year for conflict detection functions (below reduced service mode) and about 2.5 minutes per year for planning and display data recording functions (below full service mode).

Two 42-month Design Competition contracts of $200 million each were awarded in August, 1984 to Hughes Aircraft Co. and IBM Federal Systems Division, and a $4.8 billion contract was awarded to the winner - IBM - in August, 1988.

The IBM design employs IBM RISC System/6000 (RS/6000) processors as the basic subsystem interconnected by a redundant token ring local communication network. Fault tolerance is implemented for groups of processors by software-implemented Group Service Availability Management, while the highest level supervision is exercised by the Global Availability Management Service, residing on a group of up to four servers [23]. The classes of faults to be tolerated were defined as those that cause processor crash (fail-silent), omission (non-response), or performance (lateness) failures.

Fault tolerance in the RS/6000 exists for the memory (ECC, bit steering, scrubbing) and data buses (parity). The chip set of the CPU is limited to data path parity and a built-in self test sequence after power up [24]. The exclusion of faults that cause timely, but incorrect outputs is a debatable issue. The "fail-silent" assumption seems to imply a coverage of 1.0 within the processor with respect to all incorrect outputs, yet the off-the-shelf RS/6000 offers no such claim.

The largest facility of the AAS is the Area Control Computer Complex (ACCC) which may contain up to 400 processors in a distributed configuration. 23 such facilities were needed for the entire U.S.A. The ACCC design clearly was a crucial test of the usability of off-the-shelf processors and the upward scalability of the Group Service Availability Management concept.

Most regrettably, on June 3, 1994 the FAA announced the cancellation of the ACCC and of the smaller Terminal AAS system procurement. The building of the less complex Tower CCC was to be continued, and the software for the Initial Sector Suite System that had already cost over $1 billion was to be "analyzed to determine whether it can be operated and maintained." At the time this is written, definite information is not yet available as to why the ACCC was determined to be unsuitable after 10 years of design and building efforts.

It is disappointing to report the failure of the world's most ambitious and costly attempt to build the highly available ACCC system using off-the-shelf processors and software-implemented fault tolerance attributes. We only may hope that sufficient information on the successes and failures of the fault tolerance implementation of the ACCC will become available for the benefit of the builders of future systems.

6. A Model for the Future

It is evident that the complexity of today's computing and communication systems is already, in a general sense, comparable to the complexity of living organisms. However, a significant difference is noticeable in the specifications. While living organisms carry out the requirement "survive and reproduce," the requirements of the ACCC, for example, fill hundreds of pages with text, diagrams, and equations.

The evolution of living organisms and computing systems began at opposite ends. For living organisms, survival of individuals and species came first, and higher cognitive functions evolved gradually over billions of years, culminating with the emergence of *homo sapiens*. On the other hand, modern man built the computer to emulate his intellectual functions, and only grudgingly paid attention to assurance of the survival of the computer's programs in the presence of faults.

The concept of fault tolerance gathered diverse techniques that were used to cope with the consequences of faults into a cohesive set. Since then, fault tolerance has evolved as the survival attribute of computing and communication systems, especially of those in critically important applications.

As the years go by, one fundamental principle is becoming more and more evident: the more good our sophisticated computing and communication systems can contribute to the wellbeing and the quality of life of the human race, the more harm they can cause when they fail to perform their functions, or perform them incorrectly. Let us just consider the control of air, rail, and subway traffic, the emergency response systems of our cities, the flight controls of airliners, the safety systems of nuclear power plants, and most of all, the rapidly growing dependence of health care delivery on high-performance computing and communications. And the list goes on and on...

At the same time, the challenges to dependable operation are progressively growing in scope and severity. Complex systems suffer stability problems due to unforeseen interactions of overlapping fault events and mismatches of defense mechanisms. "Hackers" and individuals with criminal intent invade systems and cause disruptions, misuse, and damage. Accidents lead to the severing of communications links that serve entire regions. Finally, it may be foreseen that "info-terrorists" will attempt to cause similar damage with malicious intent.

Fault tolerance is the only guarantee that those vitally important systems will not, figuratively speaking, turn against their builders and users by failing to serve as expected because of physical, design, or human-machine interaction faults, or even because of malicious attempts to disrupt their essential services. Past experience has shown that fault tolerance is most effective when it is an integral function of every subsystem as well as a hierarchically organized function of the entire system.

Yet, it is alarming to observe that the explosive growth of complexity, speed, and performance of single-chip processors has not been paralleled by the inclusion of more on-chip error detection and recovery features. It is likely that the incorporation of these most advanced, high performance processors into high-availability life-critical systems would pose problems similar to those encountered by the AAS: can the outputs be trusted to be error-free?

One solution is the design of multiple channel systems, possibly with diverse processors. While it is suitable for modest-sized dedicated applications, such as flight control, the cost and complexity would be unacceptable for AAS or PACS-like systems and their even more complex successors in the 21st century.

The urgent problem remains: we need to convince the builders of chips as well as the buyers of AAS-like systems that the elegant ideas of building reliable systems from unreliable components that were conceived for components such as a NAND gate [6] or a relay contact [7] do not scale up to distributed systems in which each one of hundreds or thousands of component nodes is a high-performance computer itself. Redundancy alone is not the answer.

The solution that I propose here is that the model of a living organism, in which distributed, specialized survival functions protect the capacity for cognitive action, offers the best analogy for the very complex and highly dependable distributed systems that we are expected to build for the benefit (or entertainment) of our fellow humans in the near future.

Why is this model for fault-tolerant systems attractive? First, we are setting up an analogy with the most dependable information processing systems in existence - the various species of highly evolved living creatures that have survived over millions of years on our continually changing planet. There is much to be learned there, and it is not fault avoidance.

Second, the analogy reaches out to appeal to a wide spectrum of intelligent people, since it does not require knowledge of computer science or technology, or an *a priori* belief in the utility of fault tolerance.

Third, the information age is only beginning, and we should set our goals high if we are to succeed in our role as the protectors of the defining resource of the coming millennium. There is no doubt that the threats of disastrous failures are real and serious.

Four specific analogies that elaborate the model are suggested below. The first observation (focusing on humans) is that the immediate defense mechanisms of a living body (immune system, lymphatic system, pain sensors, healing processes, etc.) are autonomous (i.e. in "hardware" or "firmware") and do not require cognitive function ("software") support. However, higher-level protection (medication, imaging, physical therapy, surgery, etc.) can be invoked by the brain's decision. While this appears self-evident, most computing subsystems built today (microprocessors, disk drives, memory chips, etc.) lack a sufficient set of local error detection and recovery attributes, and software must be involved in managing their fault conditions. That is analogous to expecting cognition to compensate for the absence of immunity or of the sense of pain - not an effective substitution.

The second observation is that the immediate defense mechanisms discussed above are distributed and their generic services are shared by the specialized subsystems of the body. The analogous generic defense mechanisms are error detecting or correcting codes, totally self-checking logic, comparisons, local rollback or voting, etc.

The third observation is that diversity is the key attribute of a species that protects it against extinction due to genetic defects in the individual members, while the diversity among species has assured the continuity of life on earth. The self-evident analogy here is the use of hardware and software design diversity in fault-tolerant systems.

Finally, it may be useful to view the dependability aspects of networks of computing systems as similar to those of social structures, in which maintaining a consistent view of common reality and avoiding the distribution of (accidentally or deliberately) contaminated information are major objectives. The analogous techniques are the protocols for fault-tolerant consensus and robust data structures for data integrity.

The most immediate objective of the analogies identified above is to communicate the nature and the advantages of fault tolerance attributes to the community of users with the need for dependable communications and computing.

Customer demand is the force that will motivate the manufacturers of microprocessors and other subsystems to take a pause in their desperate race for more function, speed, and storage capacity in order to introduce at chip and other subsystem level the fault tolerance features that will allow the building of dependable systems that do not depend almost exclusively on software services to assure dependable operation.

In conclusion, I will venture the prediction that while the speed of computing will be ultimately limited by the laws of physics, the demand for more dependability will not cease as long as humans will use computers to enhance the quality of their lives.

Acknowledgement

The year 1995 marks 40 years since the author's first professional work on system dependability at JPL. It has been my good fortune to work with and learn from many colleagues at the University of Illinois, Caltech's Jet Propulsion Laboratory and the UCLA Computer Science Department. University of Illinois professors James E. Robertson and David E. Muller were my mentors and role models for my entire career.

The work with my friends in the IEEE CS Technical Committee on Fault-Tolerant Computing and IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance has been a rewarding experience and a source of inspiration for this paper.

Many thanks to my son Rimas Avižienis, EECS freshman at UC Berkeley, for preparing this text with care and dedication.

References

[1] Avižienis, A., "Design of Fault-Tolerant Computers," *AFIPS Conference Proceedings, 1967 Fall Joint Computer Conference*, Vol. 31, Washington D.C.: Thompson, 1967, pp. 733-743.

[2] "Proceedings of the Second Symposium on Large-Scale Digital Calculating Machinery," Sept. 13-16, 1949, *Annals Computation Lab., Harvard University*, Vol. XVI, Cambridge, MA: Harvard University Press, 1951.

[3] *Proceedings of the Joint AIEE-IRE Computer Conference*, Dec. 10-12, 1951.

[4] "Information Processing Systems - Reliability and Requirements", *Proceedings of Eastern Joint Computer Conference*, December, 1953.

[5] Session 14: "Symposium: Diagnostic Programs and Marginal Checking for Large Scale Digital Computers," in *Convention Record of the IRE 1953*

National Convention, part 7, New York, N.Y., March 1953, pp. 48-71.

[6] von Neumann, J., "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," *Automata Studies,* C.E. Shannon and J. McCarthy editors, Annals of Math Studies No 34, Princeton, NJ: Princeton University Press, 1956, pp. 43-98.

[7] Moore, E.F., Shannon, C.E., "Reliable Circuits Using Less Reliable Relays," *Journal of the Franklin Institute 262*, No. 9 and 10, Sept. Oct. 1956, pp. 191-208 and 281-297.

[8] Avižienis, A., Rennels, D.A., "The Evolution of Fault Tolerant Computing at the Jet Propulsion Laboratory and at UCLA: 1960-1986," in *The Evolution of Fault-Tolerant Computing*, Springer-Verlag: Vienna and New York, 1987.

[9] Avižienis, A., Gilley, G.C., Mathur, F.P., Rennels, D.A., Rohr, J.A., Rubin, D.K., "The STAR (Self-Testing-and-Repairing) Computer: An Investigation of the Theory and Practice of Fault-Tolerant Computer Design," *IEEE Trans. on Computers*, Vol. C-20, No. 11, November 1971, pp. 1312-1321; also in *Digest of the 1971 International Symposium on Fault Tolerant Computing*, Pasadena, CA, March 1971, pp. 92-96.

[10] "TOPS Outer Planet Spacecraft," *Astronautics and Aeronautics*, Vol. 8, September, 1970.

[11] Pierce, W.H., *Failure-Tolerant Computer Design*, Academic Press: New York and London, 1965.

[12] Avižienis, A., "Architecture of Fault-Tolerant Computing Systems," *Digest of FTCS-5, the 5th International Symposium on Fault-Tolerant Computing*, Paris, June 1975. pp. 3-16.

[13] Avižienis, A., "Fault-Tolerance: The Survival Attribute of Digital Systems," *Proceedings of the IEEE,* October 1978, Vol. 66, No. 10, pp. 1109-1125.

[14] Avižienis, A., Kelly, J.P.J., "Fault Tolerance by Design Diversity: Concepts and Experiments," *Computer*, Vol. 17. No. 8, Aug. 1984, pp. 67-80.

[15] Avižienis, A., Laprie, J.C., "Dependable Computing: From Concepts to Design Diversity," *Proceedings of the IEEE,* Vol. 74, No. 5, May 1986, pp. 629-638.

[16] Avižienis, A. "A Design Paradigm for Fault Tolerant Systems," *Proceedings of the AIAA Computers in Aerospace VI Conference*, Wakefield, MA, October 1987, pp. 52-57.

[17] Avižienis, A., "Software Fault Tolerance," in *Information Processing 89, Proceedings of the IFIP 11th World Computer Congress*, San Francisco, CA, G.X. Ritter (Ed.), Elsevier Science Publishers, B.V. North Holland, 1989, pp. 491-498.

[18] Joseph, M.K., Avižienis, A., "A Fault Tolerance Approach to Computer Viruses," *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, Oakland, CA, April 18-21, 1988, pp. 52-58.

[19] Avižienis, A., Ball, D.E., "On the Development of a Highly Dependable and Fault Tolerant Air Traffic Control System", *Computer*, Vol. 20, No. 2, February 1987, pp. 84-90.

[20] Avižienis, A., "The Dependability Problem: Introduction and Verification of Fault Tolerance for a Very Complex System," *Proceedings of the 1987 Fall Joint Computer Conference*, Dallas, Texas, October 1987, pp. 89-93.

[21] Avižienis, A., Huang, L.J., He, Y., Valentino, D.J., "Software and System Engineering for a Large-Scale PACS" Paper No. 2435-54 in the *Proceedings of SPIE Conference 2435, "PACS Design and Evaluation: Engineering and Clinical Issues,"* San Diego, CA, Feb. 26 - Mar. 2, 1995.

[22] Avižienis, A., "Fault Tolerance by Means of External Monitoring of Computer Systems," *AFIPS Conference Proceedings,* Vol. 50, May 1981, pp. 27-40.

[23] Cristian, F., Dancey, R.D., Dehn, J.D., "Fault Tolerance in the Advanced Automation System," *Digest of FTCS-20, the 20th International Symposium on Fault Tolerant Computing,* June 1990, pp. 6-17.

[24] Bakoglu, H.B., Whiteside, T., "RISC System/6000 Hardware Overview," *IBM RISC System 6000 Technology,* SA23-2619, IBM Corp., 1990, pp. 8-15.