DEPENDABLE SYSTEMS OF THE FUTURE: WHAT IS STILL NEEDED?

Algirdas Avižienis
Vytautas Magnus University, Kaunas, Lithuania and University of California, Los Angeles, USA — aviz@adm.vdu.lt

Abstract: It is concluded that hardware is not being adequately employed to provide system fault tolerance. A design principle called the "immune system paradigm" is presented and a hardware-implemented fault tolerance infrastructure is proposed as the means to use hardware more effectively in building dependable systems of the future.

Key words: dependable systems; immune system paradigm; fault tolerance infrastructure

1. WHAT IS THIS ALL ABOUT?

Predicting the future is not a dependable activity when breakthroughs and inventions are forecast; however, when a missing link in the chain of evolution is identified, the dependability of the prediction is likely to be more trustworthy. At least, that is my hope in presenting my thoughts on one link that I consider to be essential in building truly dependable systems in the future.

We find an interesting paradox in contemporary computing systems: they are used to provide protection for various critical infrastructures of our society, such as electrical power, transportation, communication, etc., but they do not possess an identifiable protective infrastructure of their own. Such an infrastructure should be (1) generic, that is, suitable for a variety of "client" computing systems, (2) transparent to the client's software, but able to communicate with it, (3) compatible with and able to support other defenses that the client system employs, and (4) fully self-protected by fault

tolerance, immune to the client's faults, including design faults, and to attacks by malicious software.

It is my goal to suggest what a protective infrastructure with the above properties should be like and to illustrate the concept with an architecture first presented at DSN 2000 [1] and its generalization to a hierarchy of protective infrastructures implemented entirely in hardware. I am convinced that in the first fifty years of our computer age hardware has not been adequately exploited (not even close to its full potential!) to assure system dependability, and that this omission needs to be corrected if we are going to cope with the proliferating threats to dependable and secure computing.

In the following sections, I present (1) a look at contemporary methods of protecting high-availability systems and their shortcomings, (2) the principle of design I call "the immune system paradigm", and (3) the architecture of a hierarchical fault tolerance infrastructure evolved from the infrastructure of [1].

2. FAULT TOLERANCE IN CONTEMPORARY SYSTEMS

Fault tolerance defenses in current high performance and availability platforms (servers, communication processors, etc.) are found at four different hardware levels: component (chip, cartridge, etc.), board, platform (chassis), and cluster of platforms.

2.1 Component Level Fault Tolerance

At the component level, the error detection and recovery mechanisms are part of the component's hardware and firmware. In general, we find very limited error detection and poor error containment in COTS processors. For example, in Pentium and P6 processors error detection by parity only covers the caches and busses, except for the data bus which has an error correcting code, as does the main memory. All the complex logic of arithmetic and instruction processing remains unchecked. Recovery choices are "reset" actions of varying severity. The cancellation in April 1998 of the duplex "FRC" mode of operation eliminated most of the error containment boundary. All internal error detection and recovery logic remains entirely unchecked as well [2,3].

Similar error coverage and containment deficiencies are found in the high-end processors of most other manufacturers. The exceptions are IBM S/390 G5 and G6 processors that internally duplicate the arithmetic and instruction handling units and provide extensive error detection or correction and transient recovery for the entire processor [4]. Near 100% error

detection and containment are attained in the G5 and G6 processors, which carry on the legacy of fault tolerance from the IBM 9020 – the last TCM mainframe.

Intel also has a record of building fault-tolerant systems [5]. The original design of the Pentium and P6 family of processors (except P4) includes the FRC (functional redundancy checking) mode of operation. In FRC two processors operate as a master/checker pair that receives the same inputs. The checker internally compares its outputs to those of the master and issues the FRCERR signal when a disagreement is found. The master enters the Machine Check state upon receiving FRCERR [6]. Operation in the FRC mode provides near 100% error detection at the master's output and also makes the output an error containment boundary. In April 1998 a set of specification changes was issued by Intel: "The XXX processor will not use the FRCERR pin. All references to these pins will be removed from the specification." (The XXX stands for all processors from Pentium to Pentium III). Deletion of the FRC mode left the Pentium and P6 processors with very limited error detection and containment. No further explanation was provided by Intel for the change. Our conjecture is that asynchronous inputs could not be properly handled in the FRC mode. Intel did not reply to our inquiry about the cause of FRC deletion.

Processors that do not have adequate error detection and containment can be made fault-tolerant by forming a self-checking pair with comparison (e.g., the FRC mode) or a triplet with majority voting on the outputs. Since the FRC mode deletion, there is a second deficiency of contemporary processors: they do not (or cannot) provide hardware support for comparison or voting operations.

2.2 Fault Tolerance at Higher Levels

At the board level, complete redundant hardware as well as software components are used to assure very high availability. The "hot standby" approach is especially widely used in the fields of embedded systems and telecommunications. Hot standby duplexing selectively duplicates the most critical subsystems, such as the CPU, power supplies, cooling fans, etc. Less costly fault tolerance techniques such as ECC, RAID, N+1 sparing, etc. are used for the remaining subsystems. The CPU boards present the greatest challenge: to detect faults in both CPUs and to execute a rapid switchover to the hot standby CPU when the active CPU is faulty. A good example of the state-of-the-art is the Ziatech high availability architecture [7]. The critical elements that execute CPU switchover are three hardware modules and four software modules for each CPU. These modules must be operational to make a switchover possible, but they are not protected by fault tolerance themselves.

At the platform level a widely used technique is Intelligent Platform Management (IPM) that requires the introduction of the IPM hardware subsystem into the platform [8]. It consists of additional COTS hardware (buses and controllers) and firmware that provides autonomous monitoring and recovery functions. Also provided are logging and inventory functions [8]. The effectiveness of the IPM monitoring and recovery functions is limited by the error information outputs and recovery commands of the COTS processors of the platform. For example, the P6 processors [6] have only a set of "reset" commands and five error signals (after deletion of FRCERR) whose coverage was estimated to be very limited [3].

The IPM subsystem of [8] itself is not protected by fault tolerance. The cost of adding fault tolerance may be high because of the multiple functions of the IPM. The Version 1.5 of the IPM Interface Specification (implementation independent) has 395 pages, which represent a lot of functionality to be protected. Furthermore, the IPM does not support comparison or voting for redundant multi-channel (duplex or triplex) computing.

A cluster is a group of two or more complete platforms (nodes) in a network configuration. Upon failure of one node, its workload is distributed among the remaining nodes. There are many different implementations of the generic concept of "clustering." Their common characteristic is that they are managed by cluster software such as Microsoft Cluster Service, Extreme Linux, etc. The main disadvantages for telecommunication or embedded systems are: (1) the relatively long recovery time (seconds); (2) the cost of complete replication, including power consumption, replication of peripherals, etc.

The four approaches discussed above are at different levels and can be implemented in different combinations. The integration of the different error detection, recovery and logging techniques is a major challenge when two or more approaches are combined in the same platform.

2.3 The Design Fault Problem

None of the approaches described above address the problem of tolerating design faults in hardware ("errata") and in software ("bugs") of the COTS processors. Yet a study of eight models of the Pentium and P6 processors [2] shows that by April 1999 from 45 to 101 errata had been discovered, and from 30 to 60 had remained unfixed in the latest versions ("steppings") of the processors. The discovery of errata is a continuing process. For example, consider the Pentium III [9]. The first specification update (March 1999) listed 44 errata of which 36 remained unfixed in five steppings (until May 2001). From March 1999 to May 2001 35 new errata were discovered of which 22 remained unfixed. Other manufacturers also

publish errata lists, but those of Intel are most comprehensive and well organized.

Most of the errata are triggered by rare events and are unlikely to cause system failures; yet the designers of high-dependability systems cannot ignore their existence and the fact that more errata will be discovered after a system design is complete. Continuing growth of processor complexity and the advent of new technologies indicate that the errata problem will remain and may get worse in the future.

The most effective method of design fault tolerance is design diversity, i.e., multichannel computing in which each channel employs independently designed hardware and software [10], as in the Boeing 777 Primary Flight Control Computer [11]. The Boeing and other diverse designs employ diverse COTS processors and custom hardware and software because the COTS processors do not support multichannel computing. Design fault tolerance by means of design diversity will become much less costly if it can be supported by COTS hardware elements. It is also important to note that design diversity provides support for the detection and neutralization of malicious logic [12].

2.4 Limitations of the Four Approaches

The implementation of defenses at all or some of the above described four levels has led to the market appearance of many high-availability platforms (advertised as 99.999% or better) for server, telecommunications, embedded and other applications. However, all four approaches show deficiencies that impose limits on their effectiveness.

At the component level the Intel P6 and Itanium processors, as well as those of most other manufacturers (except IBM's G5 and G6) have a low error detection and containment coverage, leaving instruction handling and arithmetic entirely unchecked. After executing the Reset command most of the existing checks (bus ECC, parity, etc) are disabled and must be enabled by software that sets bits in the (unprotected) Power-On Configuration register. In general, critical recovery decisions are handed off to software.

At the board level, such as in "hot standby" [7], unprotected "hard core" hardware and software elements handle the critical switchover procedure.

At the platform level the Intelligent Platform Management (IPM) hardware subsystem handles both critical recovery decisions and voluminous logging, configuration record keeping and communication management operations. The critical IPM element is the Baseboard Management Controller (BMC) [8] that is itself unprotected and depends on interaction with software to carry out its functions.

At the cluster level that is software-controlled the disadvantages are long recovery times and the high cost of complete system (chassis) replication.

In summary, the weaknesses are:

1. The presence of unprotected "hard core" elements, especially in the error detection and recovery management hardware and software;
2. The commingling of hardware and software defenses: both must succeed in order to attain recovery;
3. The absence of built-in support for multiple-channel computing that provides high coverage and containment, especially when design diversity is employed to attain design fault tolerance.

It is my conclusion that during the explosive evolution of hardware over the past 25 years, computer hardware has not been adequately utilized for the assurance of system dependability or survivability.

3. A DESIGN PRINCIPLE: THE IMMUNE SYSTEM PARADIGM

At the beginning, I identified four attributes that were needed by a fault tolerance infrastructure (FTI), which I see as the "missing link" in the defenses of contemporary systems, as reviewed in the preceding section. The FTI should be generic, transparent to the client's software, compatible with defenses used by the client, and fully self-protected. It is evident that an all-hardware FTI would be most likely to meet those goals, since stored programs do not need to be protected. The FTI needs non-volatile storage for record-keeping and ROM microcode for sequencing operations

While looking for a convenient way to explain the approach to system designers as well as to their customers and interested members of the public, I noted [13] that the immune system of the human body serves in a similar manner as my proposed hardware FTI in a computing system.

To develop the argument, I use the following three analogies:

1. the body is analogous to hardware,
2. consciousness is analogous to software,
3. the immune system is analogous to the fault tolerance infrastructure.

Four fundamental attributes of the immune system are particularly relevant [14]:

1. It functions (i.e., detects and reacts to threats) continuously and autonomously, independently of consciousness.
2. Its elements (lymph nodes, other lymphoid organs, lymphocytes) are distributed throughout the body, serving all its organs.
3. It has its own communication links – the network of lymphatic vessels.
4. Its elements (cells, organs, and vessels) themselves are self-defended, redundant and in several cases diverse.

Now we can identify the properties that the FTI must have in order to justify the immune system analogy. They are as follows:

1a. The FTI consists of hardware and firmware elements only.

1b The FTI is independent of (requires no support from) any software of the client platform, but can communicate with it.

1c. The FTI supports (provides protected decision algorithms for) multichannel computing of the client platform, including diverse hardware and software channels to provide design fault tolerance for the client platform.

2 The FTI is compatible with (i.e. protects) a wide range of client platform components, including processors, memories, supporting chipsets, discs, power supplies, fans and various peripherals.

3. Elements of the FTI are distributed throughout the client platform and are interconnected by their own autonomous communication links.

4 The FTI is fully fault-tolerant itself, requiring no external support. It is not susceptible to attacks by intrusion or malicious software and is not affected by natural or design faults of the client platform.

A different and independently devised analogy of the immune system is the "Artificial Immune System" (AIS) of S. Forrest and S. A. Hofmeyr [15]. Its origins are in computer security research, where the motivating objective was protection against illegal intrusions. The analogy of the body is a local-area broadcast network, and the AIS protects it by detecting connections that are not normally observed on the LAN. Immune responses are not included in the model of the AIS, while they are the essence of the FTI.

4. ARCHITECTURE OF THE FAULT TOLERANCE INFRASTRUCTURE

The FTI is a system composed of four types of special-purpose controllers called "nodes". The nodes are ASICs (Application-Specific Integrated Circuits) that are controlled by hard-wired sequencers or by read-only microcode. The basic structure of the FTI is shown in Figure 1.

The figure does not show the redundant nodes needed for fault tolerance of the FTI itself. The C (Computing) node is a COTS processor or other hardware component of the client system being protected by the FTI. One A (Adapter) node is provided for each C node. All error signal outputs and recovery command inputs of the C node are connected to its A node. Within the FTI, all A nodes are connected to one M (Monitor) node via the M (Monitor) bus. Each A node also has a direct input (the A line) to the M node. The A nodes convey the C node error messages to the M node. They also receive recovery commands from the M node and issue them to C node

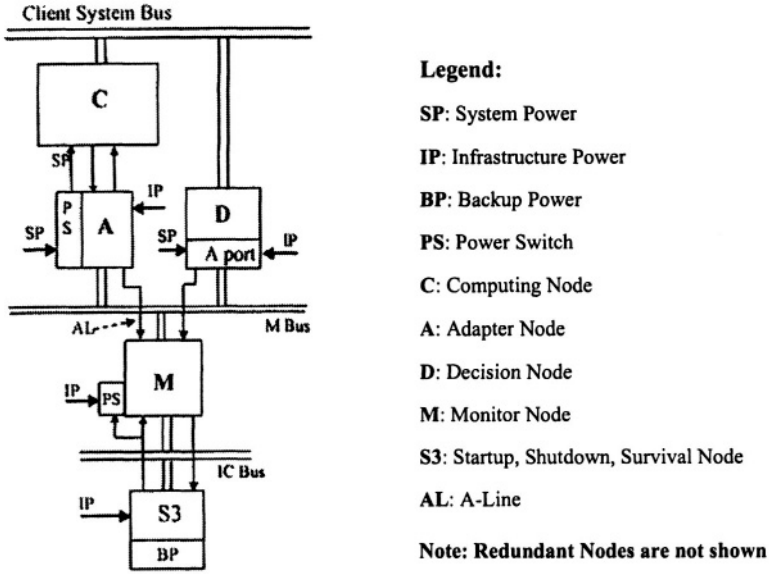

Figure 1 – The Fault Tolerance Infrastructure

inputs. The A line serves to request M node attention for an incoming error message.

The M node stores in ROM the responses to error signals from every type of C node and the sequences for its own recovery. It also stores system configuration and system time data and its own activity records. The M node is connected to the S3 (Startup, Shutdown, Survival) node.

The functions of the S3 node are to control power-on and power-off sequences for the entire system, to generate fault-tolerant clock signals and to provide non-volatile, radiation-hardened storage for system time and configuration. The S3 node has a backup power supply (a battery) and remains on at all times during the life of the FTI.

The D (Decision) node provides fault-tolerant comparison and voting services for the C nodes, including decision algorithms for N-version software executing on diverse processors (C nodes). Fast response of the D node is assured by hardware implementation of the decision algorithms. The D node also keeps a log of disagreements in the decisions. The second function of the D node is to serve as a communication link between the software of the C nodes and the M node. C nodes may request configuration and M node activity data or send power control commands. The D node has a built-in A node (the A port) that links it to the M node.

Another function of the FTI is to provide fault tolerant power management for the entire host system, including individual power switches for every C node, as shown in Figure 1. Every node exdept the S3 has a power switch. The FTI has its own fault-tolerant power supply (IP).

5. FAULT TOLERANCE OF THE FTI

The partitioning of the FTI is motivated by the need to make it fault-tolerant. The A and D nodes are self-checking pairs, since high error detection coverage is essential, while spare C and D nodes can be provided for recovery under M node control. The M node must be continuously available, therefore triplication and voting (TMR) is used, with spare M nodes added for longer life.

The S3 nodes manage M node replacement and also shut the system down in the case of catastrophic events (temporary power loss, heavy radiation, etc.). They are protected by the use of two or more self-checking pairs with backup power. S3 nodes were separated from M nodes to make the node that must survive catastrophic events as small as possible.

The all-hardware implementation of the FTI makes it safe from software bugs and external attacks. The one exception is the power management command from C to M nodes (via the D node) which could be used to shut the system down. Special protection is needed here. Hardware design faults in the FTI nodes could be handled by design diversity of self-checking pairs and of M nodes, although the logic of the nodes is very simple and their complete verification should be possible.

When interconnected, the FTI and the COTS "client" platform form a computing system that is protected against most causes of system failure. This system is called DiSTARS: Diversifiable Self Testing And Repairing System and is discussed in detail in [1]. DiSTARS is the first example of an implementation of the immune system paradigm. Much detail of implementation of the FTI is presented in the U.S. patent application disclosure "Self-Testing and -Repairing Fault Tolerance Infrastructure for Computer Systems" by A. Avižienis, filed June 19, 2001.

The use of the FTI is likely to be affordable for most computer systems, since the A, M, D, and S3 nodes have a simple internal structure, as shown in the above mentioned disclosure. It is more interesting to consider that there are some truly challenging missions that can only be justified if their computers with the FTI have very high coverage with respect to design faults and to catastrophic transients due to radiation. Furthermore, extensive sparing and efficient power management can also be provided by the FTI. Given that the MTBF of contemporary processor and memory chips is approaching 1000 years, missions that can be contemplated include the 1000-day manned mission to Mars [16] with the dependability of a 10-hour flight of a commercial airliner. Another fascinating possibility is unmanned very long life interstellar missions using a fault-tolerant relay chain of modest-cost DiSTARS type spacecraft [17]. Both missions are discussed in [1].

6. IN CONCLUSION: THE FTI WILL EVOLVE

The goal of the FTI is to use hardware more extensively and more effectively than it is being done currently in providing fault tolerance for very dependable high-performance platforms.

The DiSTARS illustration considered Intel's P6 family of processors and their supporting chipsets as the COTS elements of the host platform. These elements were not designed to utilize the FTI, which is introduced by a "retrofit."

When the FTI is provided, the processors (C nodes) can be designed to contain Adapter (A) ports, similar to those of the D node of DiSTARS. The FTI becomes simpler (the A nodes are not needed) and the FTI is used to the fullest extent when COTS processors are designed with the FTI in mind. It is also quite conceivable that the D node can be built into the processor as a "D port" as well. That leaves only the Monitor and S3 nodes as separate and independent components that are needed to implement the FTI.

A further benefit of the FTI's existence is the ability to simplify higher-level defenses that require software participation. The presence of an effective FTI simplifies the error detection and recovery requirements for system software. Currently the functions of the FTI are carried out by software, such as the Machine Check Handler in P6 family processors [6].

It is very reasonable to predict that adoption of the FTI in platform designs will lead to a better structured and more cost-effective overall dependability assurance architecture, since the other levels of protection will be supported by hardware that is missing in today's designs.

A much more drastic evolutionary step of the FTI concept is the introduction of a hierarchy of FTIs. The FTI described in [1] served a collection of COTS components located in reasonable proximity and communicating via system busses. Most likely it would be the FTIb for one board or blade. However, an FTI structure can also be incorporated in the processor chip itself. The on-chip FTIp locally manages recovery, sparing, and power management. Its Mp nodes and S3p nodes serve as the A port of the chip and communicate with the Mb nodes of the board-level FTIb. The on-chip S3p node would be simplified, since backup power is not needed at the chip level.

The next extension of the hierarchy is to install a chassis-level FTIc, where the chassis contains several boards, each with its own FTIb that contains Db, Mb and S3b nodes. The Mb and S3b nodes now serve as the A port of the board and communicate with the Mc node of the FTIc. Considering Figure 1, C is now a board and A is its Mb, D is Dc, M is Mc and S3 is S3c of the chassis, located on their own board or on one of the other boards. The S3c node has the highest shutdown and startup authority, and the S3b nodes may be simplified. The C node can also be a monitor,

printer or any other peripheral; they either have an A port built in, or an A node has to be custom designed for communication with the Mc.

It is important to note that the individual A lines, M buses and IC buses for FTI communications must be provided at each level of the hierarchy. For this reason a direct extension of the hierarchy from chassis to clusters and local networks is more problematic, but feasible. The question that remains to be resolved here is whether a single chassis can be given the authority to command shutdowns of other members of the cluster or LAN, or whether only status information will be exchanged between the Mc nodes, which then act independently. A detailed study of the hierarchical structuring is currently in progress.

ACKNOWLEDGMENTS

Ever since I wrote my first program for the ILLIAC I in 1954 and worked on the first dependability problem at JPL in 1955-56, I have been most lucky to work with and to learn from true friends: mentors, colleagues, and young researchers chasing the Ph. D. prize.

University of Illinois professors James E. Robertson and David E. Muller were mentors and role models for my academic career, and working with Ed McCluskey in the old Computer Group, later Society, taught me how to get good people to work for free and be happy about it.

The University of Illinois, Caltech's Jet Propulsion Laboratory, and the UCLA Computer Science Department will remain in my memories as places of hard work, accomplishment, and the great pleasure of shared achievement. To each one of my 31 Ph.D. winners – thanks for the challenge of keeping up with your efforts, and the same thanks go to my "UCLA club" of visiting scholars from Europe, Japan, and China who truly enriched our environment.

A most rewarding experience has been the hard work and play with my friends in the IEEE Computer Society's TC on Fault-Tolerant Computing and in the IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance. Many thanks for your dedication and friendship in working toward our common goals.

The fall of the Iron Curtain allowed me the unexpected privilege to return to my native Lithuania in 1990 and to lead as Rector the revival of Vytautas Magnus University (closed by the Soviet regime) in the city of my birth – Kaunas. Starting with 180 students in 1989, VMU now has an enrolment of over 8000, including over 1000 graduate students. Many thanks to my friends and colleagues in several EU countries for inviting me – and VMU – to take part in efforts to find a place for dependable computing in the developing European Research Area.

Finally, and most importantly – thanks to my wife Jurate and my sons Rimas and Audrius for patiently and lovingly sharing the sometimes frantic life of a restless academic competitor.

REFERENCES

[1] A. Avižienis, A fault tolerance infrastructure for dependable computing with high performance COTS components, *Proc. of the Int. Conference on Dependable Systems and Networks (DSN 2000),* New York, June 2000, pages 492-500.

[2] A. Avižienis and Y. He, Microprocessor entomology: A taxonomy of design faults in COTS microprocessors. In J. Rushby and C.B. Weinstock, editors, Dependable Computing for Critical Applications 7, IEEE Computer Society Press, 1999, pp. 3-23.

[3] Y. He and A. Avižienis. Assessment of the applicability of COTS microprocessors in high confidence computing systems: A case study. *Proceedings of ICDSN 2000,* June 2000.

[4] T.J. Slegel et al. IBM's S/390 G5 microprocessor design. *IEEE Micro,* 19(2):12-23, March/April 1999.

[5] D. Johnson. The Intel 432: a VLSI architecture for fault-tolerant computer systems. Computer 17(8):40-49, August 1984.

[6] Intel Corp. P6 Family of Processors Hardware Developer's Manual, September 1998. Order No. 244001-001.

[7] Ziatech Corp. Redundant CPU Architecture for High Availability Systems. "White paper," October 1999, available at www.ziatech.com.

[8] Intel Corp. The Pentium II Xeon Processor Server Platform System Management Guide, June 1998. Order No. 243835-001.

[9] Intel Corp. Intel Pentium III Processor Specification Update, May 2001. Order No. 244453-029.

[10] A. Avižienis and J.P.J. Kelly. Fault tolerance by design diversity: concepts and experiments. *Computer,* 17(8):67-80, August 1984.

[11] Y.C. Yeh. Dependability of the 777 primary flight control system, in R.K. Iyer et al., editors, *Dependable Computing for Critical Applications 5,* IEEE Computer Society Press, 1997, pp. 3-17.

[12] M.K. Joseph and A. Avižienis. A fault tolerance approach to computer viruses. *Proc. Of the 1988 IEEE Symposium on Security and Privacy,* April 1988, pp.52-58.

[13] A. Avižienis. Toward systematic design of fault-tolerant systems. *Computer,* 30(4):51-58, April 1997.

[14] G.J.V. Nossal. Life, death and the immune system. *Scientific American,* 269(33)52-62, September 1993.

[15] S.A. Hofmeyr and S. Forrest. Immunity by design: An artificial immune system. *Proc. 1999 Genetic and Evolutionary Computation Conference,* pages 1289-1296. Morgan-Kaufmann, 1999.

[16] Special report: sending astronauts to Mars. *Scientific American,* 282(3):40-63, March 2000.

[17] A. Avižienis. The hundred year spacecraft. *Proceedings of the 1st NASA/DoD Workshop on Evolvable Hardware,* Pasadena, CA, July 1999, pp.233-239.