

The Use of Carrier Grade Linux in Space

Kevin Scharpf
The PTR Group, Inc.
703-309-3586; Ashburn, VA,
kevin@theptrgroup.com

Dave Bryan
The PTR Group, Inc.
703-627-1735; Ashburn, VA
dave@theptrgroup.com

Mike Anderson
The PTR Group, Inc.
703-430-3748; Ashburn, VA
mike@theptrgroup.com

ABSTRACT

The telecommunications industry is embracing Linux as a means of providing high-availability, high-reliability systems. Realizing that proprietary systems have tremendous development and support costs, the push for commercial-off-the-shelf solutions that provide greater than 5-Nines reliability (no more than 5 minutes of down-time/year) is a major industry focus. The carrier grade efforts encapsulate an entire ecosystem of hardware standards for interconnection, monitoring and control as well as the software to support it. This paper examines the current state of the art in carrier grade Linux software solutions, such as those put forth by standards organizations like the Linux Foundation and the Service Availability Forum, and identify those standards and approaches that have applicability in satellite systems.

INTRODUCTION

Spacecraft designers are faced with significant challenges in the management of hardware and software faults. Hardware or software failures in space cannot be corrected through manual intervention, so the faults must either be tolerated or avoided. In addition, the vagaries of space environments pose significant hurdles for the designer.

Radiation can lead to instantaneous part failure or result in the flipping of a single bit in memory. Additionally space debris collisions may damage spacecraft subsystems rendering hardware non-functional. To further complicate the design process, even though custom-built hardware processing solutions exist to address some of these issues, these proprietary solutions lag their commercial, off-the-shelf (COTS) counterparts in speed and have significantly higher cost.

Carrier Grade Linux (CGL) grew out of an effort in the telecommunications industry where vendors found

themselves under schedule pressure to produce powerful new products while needing to maintain extremely high availability requirements¹. Like traditional spacecraft systems, the telecommunications industry was also a haven for proprietary operating systems, applications software and hardware.

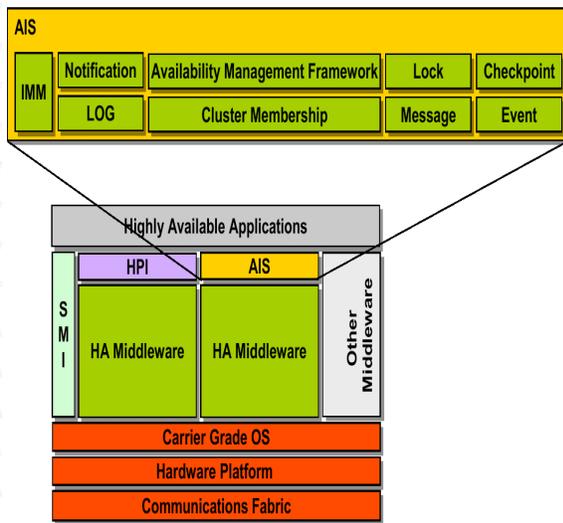
However, in the past few years all of this has changed. In order to keep up with performance and reliability requirements, the industry was essentially forced to develop standardized hardware, software interfaces and services, and protocol stacks. Much of this effort is focused on using enhanced, fault-tolerant versions of the Linux operating system.

This paper presents a high-level design for a fault-tolerant avionics system built upon Carrier Grade Linux. Using standard CGL fault detection and handling services, designers can examine the use of previously unconsidered low mass and very power efficient COTS equipment for space applications. Such an avionics system may provide a cost-effective alternative to radiation-hardened computers for LEO

and Application Interface Specification (AIS) accordingly to meet the clustering requirements for carrier-grade systems. HPI provides a common interface to CG health and status hardware. And, AIS provides a set of high availability (HA) services to be accessed and configured by HA applications.

Hardware vendors have responded by providing HPI implementations with their platforms. For those platforms whose vendors have not developed HPI implementations, the open-source OpenHPI reference implementation can be used or customized. Likewise, OpenAIS provides an open-source, production-quality implementation of the AIS specification. By developing against the AIS and HPI standards, applications are portable across a variety of hardware platforms. Figure 2 below from the Service Availability Forum illustrates the layers of the carrier grade infrastructure.

SA Forum Application Interface Specifications



A specification that defines the interface between the applications and the HA middleware, making each independent of the other.

3 Copyright© 2008 Service Availability™ Forum, Inc. - Other names and brands are properties of their respective owners. **SERVICE AVAILABILITY FORUM**

Figure 2: SA Forum Application Interface Specifications³

The Open Software Development Laboratory (OSDL), now part of the Linux Foundation, develops and maintains the Carrier Grade Linux Requirements

Scharpf

Definition. CGL incorporates the Linux operating system with carrier grade enhancements. The CGL requirements encompass the AIS and HPI standards put forth by the SA Forum. CGL requirements include serviceability, availability, security, clustering, and performance specifications. All of these requirements have their analogs in the spacecraft design business.

THE CHALLENGES OF SPACECRAFT SOFTWARE DEVELOPMENT

Two components drive the complexity of spacecraft software development: the prevalent use of custom hardware (due to power, mass, and form factor concerns) and high radiation environments which impact the OS and application software.

Custom Hardware and Software

Although spacecraft are frequently one-off systems built around custom hardware, the basic systems are typically quite similar. For example, most spacecraft implement some form of command and data handling subsystem around a particular distributed messaging API. Aerospace companies often develop in-house APIs and solutions for such functionality, but due to their proprietary nature, the implementation cannot compare to industry-wide standards. The use and deployment of systems based on open standards, such as CGL, means a significant amount of software reuse is possible between satellite programs even with different boards and interconnects.

Radiation Effects

Spacecraft software development is affected by two aspects of the radiation environment: Total Ionizing Dose (TID) and Single Event Effects (SEE). TID is a measurement of the total amount of energy absorbed over time by a device. Using the total dose tolerance of the spacecraft components and the expected radiation environment, the spacecraft designer can estimate the lifetime of components in orbit.

Radiation-hardened processors typically have a much higher total dose tolerance than COTS processors. However, for altitudes below 1400km, COTS processors are sufficiently tolerant for a 15 year mission⁴. Shielding is also effective as a means to increase the total dose tolerance. For the remainder of this paper, radiation discussions will focus on SEE effects.

Conversely, an SEE is an event produced by a single highly charged particle. The most common form of SEE is known as the SEU (Single Event Upset). The SEU most often goes unnoticed but may produce soft-

failures. Avoidance and/or tolerance of radiation effects are among the most difficult challenges faced by a spacecraft software team.

Effects of radiation on COTS and radiation-hardened boards

A typical PowerPC 750FX single board computer operates at 800MHz and can contain up to 1GB of RAM. But the processor is quite susceptible to radiation caused upsets. The COTS PowerPC 750FX processor is rated to expect 34 uncorrected errors per year at the solar minimum Galactic Cosmic Ray background rate and 320 upsets per flare for the JPL Design Case Flare⁵.

Radiation faults may be avoided by using radiation hardened components. Radiation hardening isolates electronic components in order to completely prevent latchup conditions (Single Event Latchup, or SEL) and reduces the probability of an SEU⁶. A radiation-hardened variant of the PowerPC 750 processor powers the RAD750 single board computer. The RAD750 executes at clock speeds of 133 MHz – 166 MHz and contains 128MB of EDAC RAM. The advertised SEU rate for the card is less than 1 error in 250 years⁷.

Because of the low error rate of a radiation hardened processor, the application developer may not be required to consider radiation effects within his or her software design and implementation. However, the use of radiation hardened processors carries significant drawbacks including lead-time, cost, and performance.

Fault Tolerance

A fault tolerant system assumes that faults within individual components will occur but is designed to detect and handle such faults to ensure overall mission success. The key concept with fault tolerance is the use of redundancy to detect and correct faults. Here we will look at the role CGL can play in a redundant system to assure overall mission success.

CARRIER GRADE LINUX SERVICES FOR SPACECRAFT

Carrier Grade Linux provides, through the SAForum’s AIS and HPI standards, a suite of services that can assist in the development of robust spacecraft software systems. The AIS core level services are listed below in Table 1.

Table 1: SAForum AIS Core Services

Service	Functions	Space Applicability
Availability Management Framework (AMF)	Fault detection, fault recovery, cluster coordination	Yes
Cluster Management Service (CLM)	Infrastructure, Publish-Subscribe – Node joins cluster, node leaves cluster	Yes
Checkpoint (CKPT)	Assists with fault recovery – reduce downtime	Yes
Event (EVT)	Infrastructure, Publish-Subscribe state changes	Possibly
Lock (LCK)	Infrastructure, Shared resource control	Possibly
Messaging (MSG)	Infrastructure, Inter-node, Intra-node messaging	Possibly
Information Model Management Service (IMMS)	Standard model of cluster entities	No
Logging (LOG)	Infrastructure	Possibly
Notification (NTF)	Infrastructure, failure notification to external entities	Possibly

Availability Management

The Availability Management Framework (AMF) provides services and coordination to the entire membership of a cluster. It enables high availability by monitoring and detecting the health and readiness of redundant resources in the cluster.

AMF provides a variety of means for the determination of a failed node. These means include internal

monitoring, external monitoring, and passive monitoring. Internal monitoring uses some application state to determine if the processor has failed. In external monitoring, an application does nothing toward monitoring, but its interaction with other entities or resources is monitored. An example of external monitoring would be link-level or application-level communication checks between neighbor nodes. Finally, passive monitoring is similar to a processor watchdog using features of the operating system to indicate “aliveness” of a process.

With the cluster-wide knowledge of each node state, the AMF is responsible for coordinating and providing automatic failover for failed nodes in the system. In addition, the AMF is responsible for protecting the system and system resources from a failed node. Once node failure has been detected, Carrier Grade Linux requires that the failed node is unable to access or corrupt shared resources¹². For an example of why this additional requirement is necessary, consider the condition where a node has stopped responding to a ping request. This failure is not necessarily caused by a halting failure of the failed node. If the failed node simply lacked the available processor cycles to respond, the node could wake up eventually and write old data to shared system resources. This action could corrupt the system resource. Instead AMF generally prevents such corruption by resetting the errant node. This technique is generally known as Node Fencing⁹.

Cluster Membership

The Cluster Membership Service (CLM) maintains and reports the list of nodes in the system. The list is updated anytime a node is added to or removed from the system. Any application may subscribe to CLM services to determine the active nodes in the cluster.

In addition to providing the backbone for many of the AIS Core services, CLM provides the information necessary to operate many other application layer services. Services and callback functions are available for redundant systems to manage the use of primary and backup nodes and services. Consider the handling of a load shedding algorithm. Low power conditions may require redundant nodes or entire systems to be taken offline. Using CLM, all applications affected by the loss of a node will be made aware of that loss, and can act (or not act) accordingly.

Checkpoint and Recovery

Carrier Grade Linux provides reliable and fast checkpointing services. The checkpoint (CKPT)

service enables applications to record dynamic state information. If, at anytime, a system, node, or application fault is detected, the system, node, or application may be reset and resume operation from a saved checkpoint. Checkpoint data may be replicated across multiple nodes for reliability. Performance of a checkpoint solution is crucial for many applications where it would be unacceptable to read application state from disk or flash with every usage. The performance specification for Carrier Grade Linux specifies checkpoint operations sustain 500 writes per second and 500 reads per second of 2kB blocks¹².

Events and Sensor Data Alarms

Carrier Grade Linux provides an Event Service (EVT) and a Notification Service (NTF). The EVT service provides a publish-and-subscribe API to communicate events throughout the cluster. The Notification Service is similar but is intended to pass data to systems external to the cluster¹⁰. Flight software applications have a need to push events to a set of listeners. Events may include items such as mere state change “Image Captured” or items of high criticality - “Payload Input Current High”. For the former event, a listener may exist simply to schedule the next image. For the latter event, there might be three listeners. The first listener might take immediate action to open the relay controlling the payload. The second listener would record the data into long-term storage for eventual consumption by the ground station. The third listener would send the data to the ground immediately.

Messaging

Spacecraft application developers usually develop APIs to provide messaging services between distributed nodes. Carrier Grade Linux provides a messaging service (MSG) out of the box including sophisticated features such as multicast and persistence. Multicast is implemented over named message queues. Using a named message queue means that communicating applications do not require explicit knowledge of the node that will perform the requested function. This is an important feature in supporting automatic failover.

Beyond the capabilities of many trivial message queue implementations, the message queues in CGL may be persistent. With a persistent message queue, reboots or other unexpected events can be handled and the message sequence preserved. The value of a persistent message queue is in the handling of a sequence of related messages. Loss of a single message within the series of messages invalidates the entire series.

Lock Service

CGL's Lock Service (LCK) provides node level mutex's for the protection and synchronization of resources within a cluster. This service enables consistent access to data across all nodes in a system by enabling atomic access to the resource. In a redundant system with multiple means of accessing an actuator, sensor, or other system resource, it is necessary to manage that access to prevent corruption of commands, control, or data.

Resource Monitoring

Carrier Grade Linux provides a number features for monitoring resources. High-low watermark thresholds, alarms, and "leaky buckets" are among the many methods offered¹¹.

The leaky bucket resource monitoring filter is worth mentioning. The leaky bucket algorithm is a method of implementing hysteresis in a system to stabilize its response to spurious errors. Bad readings from sensors occur infrequently and it is undesirable for on-board countermeasures to take action or otherwise report failure from a bad reading. A counter may be used to store the number of occurrences of a "failed" reading from the sensor. A threshold is set on the counter rather than the sensor reading and when that threshold has exceeded, fault handling actions occur. Meanwhile, the counter is slowly decremented in order to prevent rare failed sensor readings from tripping the threshold⁸.

AVIONICS DESIGN USING CARRIER GRADE LINUX

The design put forth here assumes the use of COTS single board computers used in a redundant configuration to provide the services and systems needed for a satellite avionics system.

Among the primary components of a flight avionics system are sensor data collection and manipulation, actuator control, control algorithms, and command and data handling. Each of these components has different availability and processing requirements. Below, we discuss each these components and their needs in a redundant system to meet system design requirements.

Avionics Control Algorithms refer to the control algorithms used for such systems as solar arrays and attitude control. Avionics control algorithms tend to be synchronous and have tight timing requirements but require small amounts of memory. Faults within these algorithms may have lasting effects on computational results. The

lasting computational effects coupled with the tight timing requirements drive the need for active processor redundancy.

Sensor Data Collection and Manipulation refers to the reading of avionics sensors, prepare and transform the raw data for higher level usage, and placing the data into long-term storage. A wide variety of sensors exist for satellite avionics. Each has different requirements for availability and redundancy. Some sensors will need redundant connections, for some it will be advantageous to have redundant sensors, and finally, some may not require redundancy at all. An avionics temperature sensor will probably not have high availability requirements nor will redundancy be required although it would be simple to achieve. At the other extreme, availability of data is critical for a GPS or star tracker while sensor redundancy is often unrealistic. In this case, the system can provide redundant connectivity to the sensor so that if the primary processor is unavailable the backup processor can seamlessly takes its place.

Actuator Control refers to such activities as closing and opening electrical relays that power various resources and subsystems. Frequently, higher-level logic pulls together sensor data with the expected state according to the actuator manipulation to determine if an error has occurred. At least one processor board must have an I/O interface to a given actuator. In order to tolerate failures, a second processor board must have an I/O interface to mission critical actuators. However, in some instances, it may be dangerous to issue the same command twice.

Command and Data Handling refers to the storage and execution of ground commands and the long-term storage of spacecraft telemetry. C&DH has large memory requirements but low processing requirements. Commands may be intended for sensors or actuators or other processors. The need for issuing commands to other processors necessitates the need for redundant interconnections between nodes. The long-term storage necessitates a high degree of fault tolerance within the data.

These functional requirements may be notionally met with a four processor-board system with redundant I/O and a redundant inter-processor communications fabric. An example system is shown in Figure 3.

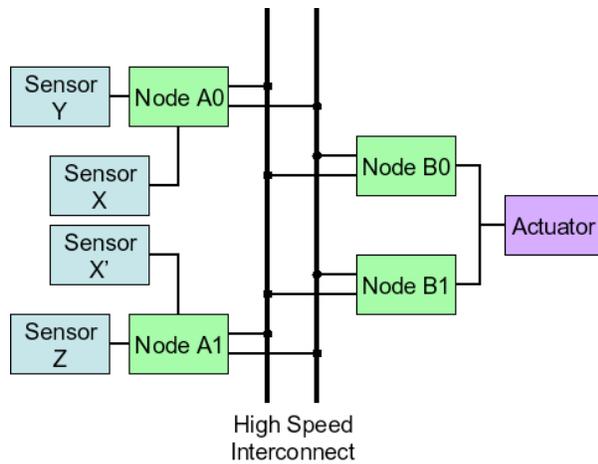


Figure 3: Redundant Avionics System

Nodes A0 and A1 demonstrate connections to redundant sensors X and X' and non-redundant sensors Y and Z. Nodes B0 and B1 demonstrate redundant connectivity to a single actuator. All four nodes are connected to two redundant communications fabrics. To provide high availability for the system, the nodes work as service pairs. While software processes can generally be executed on any of the nodes, hardware access to sensors and actuators is physically limited by the system design. Nodes B0 and B1 form a service pair for the illustrated actuator. Nodes A0 and A1 form a service pair for Sensor X and X'. The following discussion illustrates the fault tolerance techniques used in this design and how CGL enables their usage.

Detecting Radiation Events

Given that this design assumes that radiation effects will cause SEUs on a somewhat regular basis, the first step in recovery is detection of the error.

The Availability Management Framework (AMF) described earlier provides basic high-level detection of system faults. Using heartbeats and other means, this service can detect the baseline health of a node in the system. This service will also be the means by which other faults in the system can be registered with CGL so that recovery can take place.

Software EDAC

For stored commands and satellite telemetry, there is

Scharpf

significant risk of data corruption between the data write and the data read. For reference, assume 1kB data blocks, 10E-5 errors/bit-day data corruption rate, and 12-hour data storage. For this example, radiation corrupts one in twenty-five data blocks within the flight memory bank. The high error rate necessitates the usage of EDAC.

Error Detection and Correction is a means to both identify when data integrity is lost (detection) and recover the original data (correction). Data is stored with redundant bits (encoded). A common encoding technique is the Hamming Code. The Hamming Code is able to detect and correct single bit errors and detect double-bit errors.

Current research indicates that multiple-bit errors occur at a rate approximately 10% of the rate at which single-bit failures occur¹³. Furthermore, the general guidance is that the prevalence of multiple-bit errors increases as device sizes continue to shrink⁷. This fact tends to be the primary reason for the use of low density memory on spacecraft systems today.

Returning to the avionics design example, the design deploys a software EDAC algorithm. While reading and writing commands and telemetry from the long-term data bank, software EDAC is used to detect and correct single bit errors. For multiple bit errors, the node may register the fault with the system and be reset.

In addition, an EDAC “scrubbing” process running in the background provides a continuous sweep through registered memory (whether program or data) and can clean up any single bit errors, and detect multiple bit errors.

Data Handling

Sensor data collection and manipulation requires an application to read raw data from a variety of sensors, process and place the data into long-term storage, and forward the data to any requesters.

It is usually important for spacecraft applications to deploy monitors of sensor data. These monitors generate both a custom corrective action and place an event in long-term storage to indicate the failure and the action taken. CGL supports monitors in two ways.

First, CGL allows the designation of trivial thresholds associated with the raw or derived data. Threshold failures generate notifications placed in long-term storage. As these notifications support multiple

listeners, a secondary listener may exist to take corrective action upon the notification.

A second important way in which CGL assists with data handling is hysteresis. Raw data reads from sensors are notoriously noisy. CGL supports the designation of a leaky bucket algorithm where single-event errors during data reads do not affect the failure trend. The developer defines high and low water marks for the bucket and transitions past these thresholds generate failure notifications.

N-Modular Redundancy

N-Modular Redundancy (NMR) may be used to further detect faulty operations. Most implementations of these voting algorithms require specialized hardware. However, for Carrier Grade Linux, the only requirement is a high bandwidth interconnect. The existing Carrier Grade services provide the base platform for such an implementation.

To see how NMR may be implemented over Carrier Grade Linux, consider the use of multipoint-to-multipoint communications through message groups. The vote occurs in one message group while the calculation occurs in the second message group. The group to evaluate the votes consists of an active node and a passive spare node. Meanwhile, the group that performs the calculation consists of three or more active nodes. The message containing the input data is sent into the calculation group. When each node has completed the calculation, the message is sent into the vote evaluation group. The votes are counted and disagreement results in an event notification (see the Notification service in Table 1) that a particular node has failed. The Availability Management Framework handles the node failure event by restarting the failed node from the most recent checkpoint. A model of this prospective solution is illustrated in Figure 4.

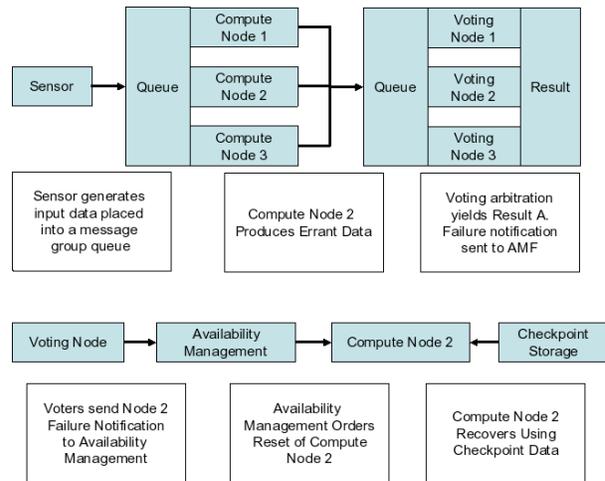


Figure 4: Illustration of NMR using CGL Services

Fault Recovery

Fault recovery clears the fault on the failed node and enables continuous operation through the reallocation of system resources. In order to clear the fault, the failed node is typically reset. Spare nodes are reallocated to perform the functions originally allocated to the failed node. The spare nodes read the current application state from a checkpoint for rapid recovery.

Fault clearance

CGL supports different means to clear a fault. These means include trivial restart of a task to a reboot or power-cycle of the failed node. The developer may use CGL’s resource monitoring techniques of counters and leaky buckets to define escalating fault clearance methods.

Sparing

Though N-Modular Redundancy requires an odd number of nodes, our system design must allow for the reset of any one of the active nodes at anytime. In order to provide seamless availability to all sensors and actuator interfaces, the “spare” fourth processor must be kept on-line for immediate failover should its service partner get reset.

Referring to Figure 3 above, nodes B0 and B1 provide access to an actuator. Consider the state where A0, A1 and B1 are active nodes and B0 is a spare. If fault is detected on active node B1 and the node requires a reset, the actuator would be unavailable while B1 is brought back online. However, because B0 exists as a

hot-spares, the actuator interface remains seamlessly available.

In addition to the processors defined, it is possible to add further redundancy to the system with additional cold spares. Cold spares in a design can be used to address total dose radiation concerns. In particular, a powered off electronic device is less susceptible to ionizing radiation and will be available in the event an active device fails.

Recovery

Beyond the reset of the failed node, CGL coordinates resource reallocation and checkpoint recovery. Reference the hardware layout in Figure 3; assume a failure of node B0. B0 and B1 provide the same service, actuator control, so CGL assigns B1 to perform actuator control. Meanwhile, CGL notifies the cluster that B0 is no longer a member of the cluster. In order for B1 to assume B0's functionality with zero downtime, B1 reads B0's application state from CGL's replicated checkpoint.

Note that the design has redundant high-bandwidth interconnects. CGL system software supports redundant interconnections directly. Therefore, even a double-failure where B0 fails and B1's primary network connection fails does not lead to downtime. With the low downtime provided by native CGL services, an avionics system built upon redundancy become practical.

CONCLUSION

The level of services provided by Carrier Grade Linux enables the usage of advanced spacecraft fault tolerance techniques with little effort required by the application developer. Fault tolerance techniques such as checkpoint and recovery and data replication are available out of the box. Additional supporting infrastructures for messaging and event notifications provide useful standards-based functionality that may replace proprietary infrastructure designs. This paper has described how the services provided by CGL apply to spacecraft environments. Using these services, we have described a hypothetical cluster as a low-cost, high-availability spacecraft avionics solution.

REFERENCES

1. Vaidya, Niranjin, "Off-the-Shelf Carrier-Grade Middleware: The Next Logical Step," RTC Magazine,

September 2006.

2. PCI Industrial Computers Manufacturers Group, "PICMG 3.0 ATCA Short Form Specification", <http://picmg.org>, January 2003.

3. Service Availability Forum, "Service Availability Forum Tutorial Application Interface Specification (AIS)", <http://www.saforum.org>, 2006.

4. Lahti, Doyle, Grisbeck, Gary, Bolton, Phil, "ISC (Integrated Spacecraft Computer) Case Study of a Proven Viable Approach to Using COTS in Spaceborne Computer Systems", 14th Annual/USU Conference on Small Satellites, 2000.

5. Hillman, Robert, Conrad, Mark, Layton, Phil, Thibodeau, Chad, "Space Processor Radiation Mitigation and Validation Techniques for an 1800 MIPS Processor Board", 7th European Conference on Radiation and its Effects on Components and Systems (RADECS) 2003.

6. Mayer, Donald C. and Laco, Ronald C., "Designing Integrated Circuits to Withstand Radiation", Crosslink, The Aerospace Corporation, 2003.

7. BAE Systems, RAD 750 Detailed Brochure, December 2002.

8. EventHelix, "Software Fault Tolerance", Retrieved 3 June 2006 from EventHelix web site, <http://www.eventhelix.com/RealtimeMantra/SoftwareFaultTolerance.htm>.

9. High Availability Linux Project, "Shoot The Other Node In The Head (STONITH)", retrieved 3 June 2006 from the Linux-High Availability web site, <http://linux-ha.org/STONITH>.

10. Naseem, Asif, "An In-Depth Look at SA Forum Interface Specs", Network Systems Designline, February 2006.

11. Open Source Development Labs Carrier Grade Linux Working Group, "Carrier Grade Linux Availability Requirements Definition Version 4.0", Copyright 2005-2007.

12. Open Source Development Labs Carrier Grade Linux Working Group, "Carrier Grade Linux Clustering Requirements Definition Version 4.0", Copyright 2005-2007.

13. Ladbury, Ray, "SDRAM Error Modes—Characterization, Rate Calculation and Mitigation", Military and Aerospace Programmable Logic Device International Conference 2002.