Simplex Architecture: Meeting the Challenges of Using COTS in High-Reliability Systems

Lui Sha, John B. Goodenough, and Bill Pollak
Software Engineering Institute

Since the end of the Cold War and the downsizing of military budgets, it has been more important than ever that mission-critical systems be reliable, affordable, and capable of evolving to prevent obsolescence. Furthermore, as operational software systems play a more critical role in both military and nonmilitary applications, the need for dependability in all software systems is increasing. In this article, we review a new combination of existing technologies that can meet these challenges.

The Challenges

To cut costs and gain leverage from technical advances in the commercial sector, the Department of Defense (DoD) has actively encouraged the more frequent use of commercial-off-the-shelf (COTS) components in its software systems. This DoD mandate challenges systems developers to integrate COTS components into systems without compromising the strict reliability and availability requirements of DoD applications. What is more, there are significant strategic and tactical advantages afforded by the ability to adapt quickly to changing situations. These potential advantages challenge developers of DoD systems to find ways to modify and upgrade system components more quickly while reducing the possibility of error.

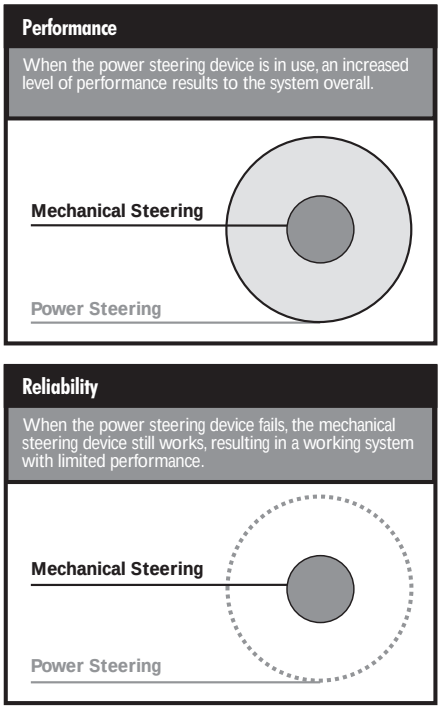
In hardware, problems inherent in the use of COTS components in harsher environments—such as those in which DoD systems operate—can often be solved by packaging. System-level hardware reliability can also be improved by the use of standard fault-tolerance technologies. For example, COTS hardware components can be replicated (replication) and a vote can be taken on their outputs (majority voting). These methods can provide significant protection from hardware faults.

The SEI's work is supported by the Department of Defense. An earlier version of this article appeared in Bridge *(August 1997), a publication of the Software Engineering Institute.*

To ensure the reliability of software is far more difficult. Statistics from the field indicate that software faults cause system failures about 10 times more often than hardware faults [1]. Although a high-assurance software development process can significantly reduce the number of software faults, such processes are typically used only for custom-made software—software designed to one customer's specifications. Most COTS software components, however, are sold as "black boxes" with no warranty and are not typically subject to rigid development, verification, or testing processes. It often is possible to obtain the source code of a COTS software component by paying a large sum of money to the vendor. With the source code, the customer can then subject the COTS components to a high-assurance inspection and testing process and make any modifications that are needed. But once a COTS software component has been modified, it is no longer COTS software, and because the modified COTS software is no longer compatible with the vendor's future releases, most if not all of the benefits of the COTS approach are lost. Therefore, this approach—making proprietary modifications to COTS components—is inconsistent with the original motivation for their use.

Existing architectures cannot tolerate software faults, including faults caused by COTS components or by component changes. This makes the DoD mandate to increase the use of COTS components a challenge to implement. Systems must maintain their existing level of performance even when upgraded components are introduced and do not work under all circumstances. For COTS components to be used safely and effectively, a software fault-tolerant architecture—one that allows developers to modify existing applications and to try out new or upgraded COTS software components easily, affordably, and reliably—is essential.

Figure 1. *Analytically redundant module: a hardware example.*



The Simplex Architecture Solution

Replication and majority voting are effective tools for dealing with random hardware faults. The probability that the majority of replicated hardware components will have the same random fault is extremely small. Unfortunately, replication and majority voting are ineffective against software faults. Given the same inputs, replicated software components will produce the same results, right or wrong.

N-version programming is an approach that is intended to randomize software errors and thus make majority voting work for software faults. In this technique, different programmers build different versions of the same software (or critical parts of a software system) with the idea that different designers and implementers will produce different errors. Therefore, when one system fails under a given set of circumstances, the other probably will not fail. A pragmatic way to use N-version programming is to use different vendors' COTS components with the same interface. For example, in the Boeing 777, three different vendors' Ada run-times and compilers are used [2]. However, because some studies have indicated that some errors will still be shared among the independently developed systems [3], the FAA DO 178B certification process requires that each of the run-times be certified together with the applications. As pointed out by FAA DO 178B, N-version programming may provide some reliability improvement, but the improvement cannot be quantified, and the results cannot be relied on.

Software faults are the result of product complexity that is beyond the developers' capabilities in specification, design, verification, and testing. A well-established engineering approach to guard against the failures of a complex system is to provide a simpler back-up system with assured reliability. For example, the power-assisted steering system in cars is built on and backed up with a simpler mechanical steering system. The two steering systems are not different designs that meet a common specification; the requirements for them are different. One set of requirements emphasizes performance (ease of steering) while the other emphasizes reliability: safe operation even in the presence of engine or hydraulic-system failure. The mechanical steering system is said to be analytically redundant with the power-assisted steering system in the sense that it provides just enough of the power steering system's performance to assure safety (see Figure 1). Power-assisted brakes follow the same principle.

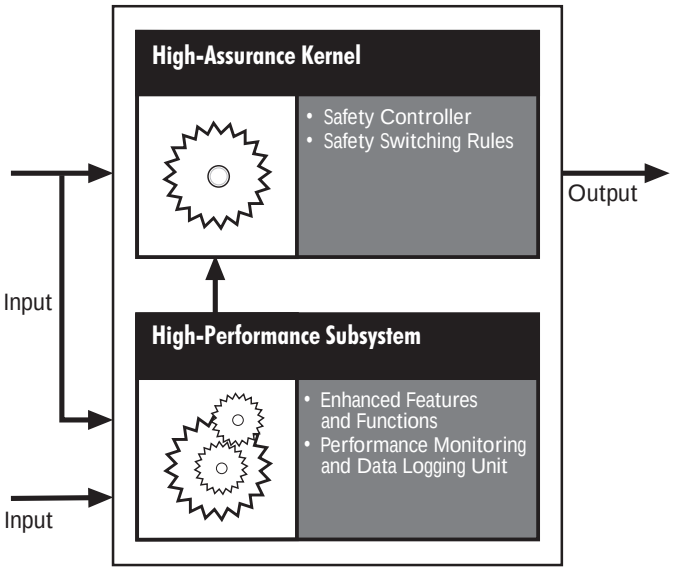
Analytic redundancy can be and has been applied to software systems. Using analytic redundancy, a system is partitioned into a high-assurance portion and a high-performance portion. The high-assurance application kernel is designed to ensure simplicity and reliability. Because of the need to apply costly high-assurance processes to the kernel, the system must be designed such that the rate of changes to the high-assurance kernel is much slower than the rate of changes to the high-performance subsystem. Therefore, COTS components with uncertain reliability are not used in the high-assurance kernel. On the other hand, COTS components *can* be used extensively in the high-performance subsystem. This model is applied in the Boeing 777: a high-assurance backup software controller, known as the secondary digital controller, implements the tried-and-true 747 control laws, whereas a high-performance 777 software controller serves as the normal digital controller [2].

To do the right control job, the high-performance subsystem receives information from a wider variety of sensors compared to the information that the high-assurance kernel receives. The kernel monitors the system state. If the subsystem is driving the system toward a state that the kernel cannot control, the kernel dynamically takes over, meaning that the kernel's outputs are used instead of those of the high-performance subsystem. The kernel can reset and restart the subsystem if and when certain constraints are violated. After the kernel has successfully brought the system back to a new and stable system state, the kernel switches control back to the high-performance subsystem (Figure 2). Since residual software errors are activated only infrequently in certain system states, the subsystem will behave correctly most of the time. This approach works well for systems that have states that can be monitored, such as feedback control and command-and-control applications.

When combined with technologies for real-time computing and component swapping, this approach can also be used to implement upgrades to the high-performance subsystem while the system is on-line. The upgrade need not be perfectly reliable. Failures in upgrades of the high-performance subsystem are no different from the activation of residual errors in the subsystem: The kernel will take over if the new subsystem misbehaves. In addition, the kernel can dynamically return control back to the old version of a component when the upgrade fails, as shown in Figure 3.

Software engineers at the Software Engineering Institute (SEI) have integrated well-established technologies—high-assurance application-kernel technology, address-space protection mechanisms, real-time scheduling algorithms, and

Figure 2. *Run-time replaceable analytically redundant unit.*



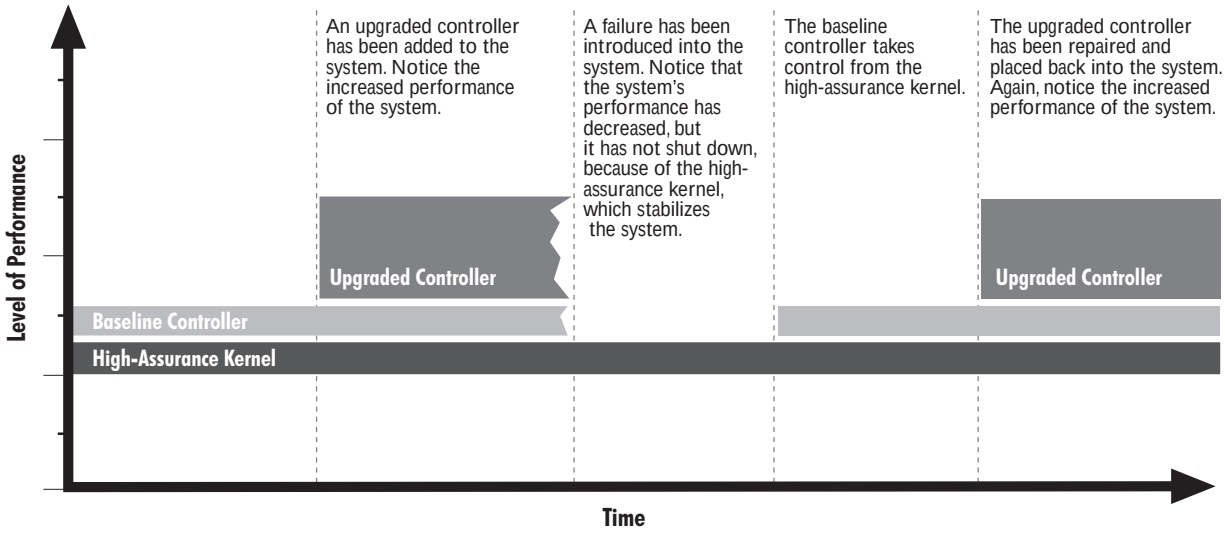


Figure 3. *Analytically redundant module: reliability and performance.*

methods for dynamic communication among modules—to create a framework for the reliable evolution of software systems. This framework is called the Simplex architecture [4]. Although most of the technologies upon which the Simplex architecture is based have existed for some time, the increased adoption of these technologies is making the Simplex architecture increasingly viable.

Under the Simplex architecture, each major system function is implemented as an analytically redundant module consisting of a high-assurance application kernel and a high-performance subsystem, the components of which can be swapped in real time. Like power-assisted steering and power-assisted brakes in a car, analytically redundant software modules can be put together to form an application just as any modules can, except that the components in an analytically redundant module can be replaced easily, reliably, and with no adverse effect on the rest of the system. Should the high-performance portion prove through deployment to be sufficiently reliable, the Simplex architecture also permits users to replace an analytically redundant module with a nonredundant software module consisting only of the high-performance portion. In this way, users can dynamically balance the sometimes conflicting concerns of reliability and efficiency.

In addition to the maintenance of system reliability when COTS software is used, Simplex has proven to be useful for other dependable-system applications. The SEI has participated in several pilot studies that have tested the concepts described in this article in prototypes of real-world applications. These include

- The INSERT (INcremental Software Evolution for Real-Time applications) project: The objective of this project is to generalize and scale up the technologies of the Simplex architecture for dependable evolution of on-board avionics systems. The problem of upgrading mission-control software for the F-16 aircraft is being used as a demonstration vehicle. The sponsors are the Defense Advanced Research Projects Agency Evolutionary Design of Complex Software Programs and the Air Force Research Laboratory (AFRL). The participants are AFRL, Lockheed Martin, Carnegie Mellon University, and the SEI.
- New Attack Submarine Program fault-tolerant submarine control: The objective of this project is to develop, demonstrate, and transition a COTS-based fault-tolerant control system that can be upgraded inexpensively and dependably. The sponsors are the Office of Naval Research (ONR) and Naval Systems Engineering Activity PMS-450. The participants are Naval Surface Warfare Center, Carderock Division, ONR, Electric Boat Corporation, and the SEI.
- CMU's Real-Time Multivariable Control of Plasma-Enhanced Chemical Vapor Deposition project [5]: The objective of this silicon wafer manufacturing project is to demonstrate the use of the Simplex architecture as a basis for the control architecture in manufacturing process-control software. The project is based on a suggestion by engineers from SEMATECH (SEmiconductor MAnufacturing TECHnology). The participants are Carnegie Mellon University and the SEI.

The SEI has also developed demonstration prototypes of the Simplex architecture. The simplest of these demonstration prototypes can be viewed as a QuickTime movie on the SEI Web site (http://www.sei.cmu.edu/technology/simplex/SIMPLEX.MOV). In this demonstration, a feedback-control-loop device controls the positioning of an inverted pendulum. The purpose of the control software is to balance the pendulum in an upright position and keep it as close to the center position as possible. The demonstration shows a safe on-line upgrade from a legacy C program to an Ada 95 program that implements an improved control algorithm. The Ada program visibly im-

proves the control performance. When a bug is introduced into the Ada code and the flawed Ada program is swapped back into the system, the system detects the fault and transfers control back to the C program. The pendulum remains in balance throughout the transfer to the high-performance Ada program, the transfer to the flawed Ada program, and the reversion to the C program. Live interactive demonstrations of more advanced applications such as distributed fault-tolerant controls are available at the SEI for those who wish to pursue this subject further.

For more information about Simplex architecture, visit the SEI Web site at http://www.sei.cmu.edu/technology/dynamic_systems/simplex/introduction/simplex01.shtml.

Summary

It is more important than ever that mission-critical systems be reliable, affordable, and capable of evolving to prevent obsolescence. In this article, we have reviewed a set of existing technologies upon which we can develop an application architecture that is designed to meet these challenges.

The early success of analytically redundant software modules in high-reliability applications provides grounds for optimism that the DoD can achieve the goal of reliable, affordable, evolutionary acquisition of mission-critical systems that exploit the advantages of COTS components. ◆

Acknowledgments

We thank Carol Sledge and John Foreman for their helpful reviews and Mark Paat for creating the graphics for this article.

About the Authors

Lui Sha is a senior member of the technical staff of the SEI, a fellow of the Institute of Electrical and Electronics Engineers (IEEE), an associate editor for the *Real-Time System Journal*, and the chairman-elect of the IEEE Real-Time Systems Technical Committee. He made key contributions to the development and transition of generalized rate monotonic scheduling theory, which has been successfully used in practice and is now supported by the POSIX real-time extension and by Ada 95. He formulated the Simplex architecture and led its development from a concept to the core product of the SEI's Dependable Systems Upgrade Initiative.

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
Voice: 412-268-5875
Fax: 412-268-5758
E-mail: lrs@sei.cmu.edu
Internet: http://www.sei.cmu.edu/technology/dynamic_systems/simplex/

John B. Goodenough is the chief technical officer of the SEI and was named a fellow of the Association for Computing Machinery in 1995. He is the former leader of the Rate Monotonic Analysis for Real-Time Systems Project. He was a distinguished reviewer for the Ada 95 language revision effort and has served as head of the U.S. delegation to the International Organization for Standardization working group on Ada. He was the principal author of the document specifying the revision requirements for Ada 95 and has served as chairman of the group responsible for recommending interpretations of the Ada language.

Bill Pollak is a senior writer and editor, member of the technical staff, and team leader of the Technical Communication team at the SEI. He is the editor and co-author of *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems* (Kluwer Academic Publishers, 1993) and has written articles for the *Journal of the Association of Computing Machinery Special Interest Group for Computer Documentation* and *IEEE Computer*.

References

1. Gray, J., "A Census of Tandem System Availability Between 1985 and 1990," *IEEE Transactions on Reliability 39*, Vol. 4, October 1990, pp. 409-418.
2. Yeh, Y.C., "Triple-Triple Redundant 777 Primary Flight Computer," *Proceedings of the 1996 IEEE Aerospace Applications Conference*, Vol. 1, New York, N.Y., Feb. 3-10, 1996, pp. 293-307.
3. Knight, J.C. and N.G. Leveson, "An Experimental Evaluation of the Assumption of Independence in Multi-Version Programming," *IEEE Transactions on Software Engineering*, SE-12(1):96-109, January 1986.
4. Sha, L., R. Rajkumar, and M. Gagliardi, "Evolving Dependable Real-Time Systems," *Proceedings of the 1996 IEEE Aerospace Applications Conference*, Vol. 1, New York, N.Y., Feb. 3-10, 1996, pp. 335-346.
5. Knight, T.J., D.W. Greve, X. Cheng, and B.H. Krogh, "Real-Time Multivariable Control of PECVD Silicon Nitride Film Properties," *IEEE Transactions on Semiconductor Manufacturing*, Vol. 10, No. 1, February 1997, pp. 137-146.