Issues Using DoDAF to Engineer Fault-Tolerant Systems of Systems

Dr. Ronald J. Leach
Howard University

It is unreasonable to expect all portions of complex, safety-critical, net-centric systems to always function properly. Hence, fault-tolerance techniques are necessary to ensure overall satisfactory system operation. This article considers some aspects of the relatively new Department of Defense Architectural Framework (DoDAF) in the systems engineering of complex systems that can be used to improve fault tolerance. It also describes some apparent deficiencies of DoDAF from a fault tolerance perspective and provides some suggestions for improvement.

There have been many advances in fault tolerance techniques since the early work of Brian Randell [1] on safe-state rollback and Algiradis Avizienis [2] on multiple code version redundancy. Several computer languages, including Ada, and operating systems have support for exception handling or fault isolation. Improved software engineering practice, especially the Software Engineering Institute's Capability Maturity Model® (CMM®) and CMM Integration℠ (CMMI®), has led to better data collection and analyses of faults and failures, thereby improving safety-critical systems [3, 4].

Unfortunately, there has been little research on the direct impact of fault tolerance techniques on systems engineering and requirements engineering of *systems of systems* (SoS), large systems that, themselves, are composed of multiple systems. The theme of this article is that there are opportunities to engineer fault tolerance into complex systems early in the life cycle, not just focusing on better ways to encode designs. The point is illustrated with a discussion of the use of DoDAF [5, 6, 7]. It is important to keep in mind that DoDAF is relatively new, with version 1.0 having been released in February 2004. Given the lead time for the development of complex systems, it is not reasonable to expect detailed analyses of DoDAF-based systems development, much less lessons learned. Some academic analyses have been performed; one example of such an analysis is [8].

Fault Tolerance, Reuse, and Commercial Off-The-Shelf (COTS)

It is well known that it is impossible to test fully all possible execution paths in large software systems. Therefore, most software is deployed with some number of software errors or faults, which may lead to system failures in the sense that a failure occurs when a system does not meet its specifications of behavior. This situation occurs regardless of whether the system is entirely created from new code or, as most systems are created today, with a heavy use of reusable components of a variety of sizes.

Since most large safety-critical systems are released and deployed with some (often unknown) number of faults, techniques that allow the effect of faults to be isolated, to be prevented from propagating from the subsystem in which they originate, or to have their effects minimized are often critical. It is especially important to mitigate any fault that may occur in COTS products whose internal structure and potential faults are often extremely difficult to determine.

> *"Fault tolerance is critical, and support for it is likely to become more important in the future."*

Some of the work of Jeffrey Voas [9, 10] has provided important advances in fault-tolerant systems – the systematic analysis and prediction of fault propagation across the boundary of subsystems, especially COTS products. This clearly illustrates that software fault tolerance is intimately connected with reuse and systems engineering. Of course, many developers of COTS products are not concerned (or deeply concerned) with fault tolerance. However, the deployers of such products within safety-critical systems must be.

Most large, complex, software systems developed in the last two decades have attempted to reuse existing software components as much as possible. Reuse has enormous potential for cost savings, but poses some issues for development of fault-tolerant systems because of unknown quality of some components, especially COTS products.

Wrappers, *bridgeware*, or *glueware* are often created to provide an interface between applications, components, data stores, or systems. Their interfaces and functionalities are easy to specify during the requirements gathering period, assuming the high-level interfaces of items that are to be *wrapped* are known. David Corman discussed technologies for wrapper generation in a 2001 CrossTalk article [11].

Unfortunately, wrappers may hide unknown defects within individual system components, since the internal interfaces and quality of COTS products are often difficult or impossible to determine.

The timing performance of systems with wrappers can be hard to determine. It can be difficult to determine if some actions are atomic (non-interruptible), which is often a requirement for system correctness; coordination of atomic actions across networks is even harder to guarantee. There are some fault-tolerance issues that are caused directly by improper use of wrappers which are discussed briefly when (Operational View) OV-6c diagrams are described later in this article.

Recovery blocks (areas of *safe* code) and redundancy are likely to be useful for functions, procedures, or objects that reside on a single system. Wrappers, bridgeware, or glueware are highly likely to have to function across two or more servers, depending on where the applications being *glued* reside. The applications themselves may require the use of several different servers. Hence, it can be difficult to find a safe state where all servers are consistent. Guaranteeing coordinated atomic actions is difficult, but essential, if fault tolerance can be achieved.

It is difficult to provide fault-tolerance for COTS products because generally

there is little knowledge of the internal interfaces of the product. Exception handling is usually done in wrappers and operating systems. Fault tolerance in operating systems can be helpful here if the COTS product is running on a single server. If the COTS product requires running on multiple servers, the situation becomes much more complex.

Systems created from COTS products often have predictable initial cost, although maintenance costs may be much larger than expected for long-lived systems, as Voas [10] and Leach [12] suggest. Additional problems occur if the schedule of releases of new versions of a COTS product is out of phase with the system to be deployed or the vendor of the COTS product stops product support during the desired system's life cycle.

The most extreme case of reusing software is new systems that are themselves composed of systems. These SoS will exhibit most of the difficulties indicated earlier, and to a much greater extent than what is typical of the *simpler* cases described previously. Systems too large to run on a single server need redundant hardware to provide continuity of operation if failure occurs; redundancy may also be required on a single-server system.

SoS will exhibit most of the difficulties indicated earlier, and, in many net-centric systems, it is impossible to allocate a set of hardware to preserve system state in the event of an unforeseen fault. Hence these systems, especially safety-critical ones, are greatest in need of systems development processes that support and encourage fault tolerance.

Systems Engineering With DoDAF

DoDAF was developed to help with the design of complex SoS. DoDAF is designed to work with several types of software development methodologies, notations, and Computer Assisted Software Engineering (CASE) tools. In this section, we discuss the gap between the results of dependability research and the incorporation of these results into the engineering of complex systems. Most dependability research focuses on lower-level actions such as exception handling, often with new languages [13]. Our DoDAF experience is with Telelogic's high-quality System Architect for DoDAF (Vers. 10.3.19), which supported all modeling requirements of a portion of a large multi-year project.

DoDAF supports the creation of several types of views, which provide insight into the complete requirements, design, and development process. The views can be classified into three broad categories: OVs, systems views (SVs), and technical standards views (TVs), each of which conveys different aspects of the architecture in several products. All views can be augmented by providing context, summary, or overview-level information or an integrated dictionary to define terms.

OVs depict what is going on in the real world that is to be supported or enabled by systems represented in the architecture. Activities performed as parts of system operations and the associated information exchanges among personnel or organizations are the primary items modeled in OV. The OV reveals requirements for capabilities and interoperability.

SVs describe existing and future systems and physical interconnections that support the needs of the originating funding organization that are documented in the OVs.

TVs are used to catalog standard system parts or components and their interconnections. This view augments the systems view with technical detail and forecasts of standard technology evolution. The terms *technical view* and *architectural view* are sometimes used instead, especially in documents before release 1.0 of the DoDAF handbook was widely available.

The potential for inclusion of fault tolerance in each type of DoDAF view is discussed in turn. For completeness, some descriptive material is included in this section with minor changes to meet space limitations; the material is taken from the public domain DoDAF materials provided by the DoD Architecture Working Group as listed in the references.

The first set of DoDAF views described here is known as OVs, which show the essence of system operations. These views are illustrated with diagrams describing the docking of the Crew Exploratory Vehicle (CEV) with the International Space Station (ISS) under the guidance of ground control.

High-Level Operational Concept (OV-1)

This is an informal, cartoon-like, graphical diagram intended for high-level presentations. Fault tolerance is not usually illustrated here. Our OV-1 example diagram (Figure 1) shows the tracking and relay satellite system interface of the ISS and CEV. The icons and connections are not entered into any internal data structures of the CASE tool and, thus, are not available to the rest of the representations.

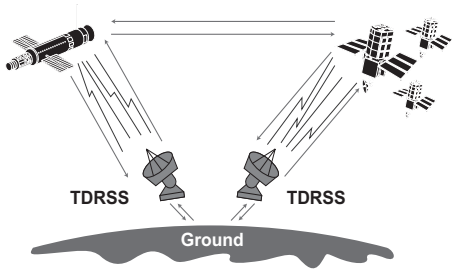


Figure 1: *High-Level Operational Concept (OV-1)*

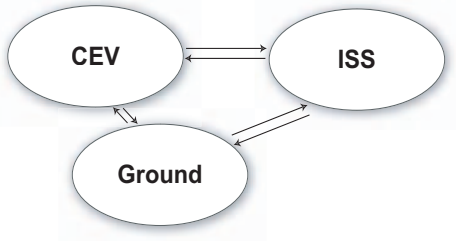


Figure 2: *Operational Nose Connectivity (OV-2)*

Operational Node Connectivity (OV-2)

Figure 2 tracks exchange of information from key operational nodes. Redundant connections to provide fault tolerance using backup systems and recovery routines can be specified here in detail using OV-2 *needlines*. The detailed information needed by the operational nodes is not displayed in this diagram in Telelogic's System Architect for DoDAF tool; such data is, however, stored internally in a database and is visible in the OV-3 matrix.

Operational Information Interchange (OV-3)

This matrix expresses relationships between the three basic OV architecture data elements: operational activities, operational nodes, and information flow. The matrix is generated automatically from the detailed information entered in the OV-2 database. This matrix has the same fault-tolerance aspects as OV-2 diagrams.

Organizational Relationships (OV-4)

This chart clarifies relationships between organizations and sub-organizations. It is not applicable to fault tolerance.

Operational Activity Model (OV-5)

Figure 3 (see page 24) delineates lines of responsibility for activities; uncovers redundancy; suggests decisions about streamlining, combining, or omitting activities; and defines or flags issues that need to be scrutinized further. It is the basis for OV-6 depictions of activity sequencing and timing. Fault tolerance can be incorporated here explicitly, since *operational control activity redundancy* (a DoDAF

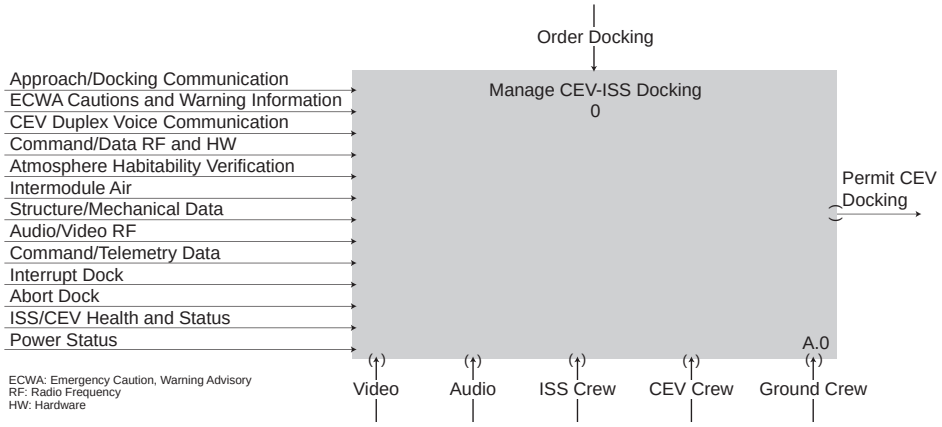


Figure 3: *Operational Activity Model (OV-5)*

term) is a primary example of fault tolerance.

An OV-5 diagram is shown for the parent operational node; there are similar ones (not shown) for each of the child nodes. The arrows are Input, Control, Output, and Mechanism (ICOM). An ICOM diagram always has the Input arrows on the left of an Operational Activity Node, Control arrows on the top, Output arrows on the right, and Mechanism arrows on the bottom. The mechanism arrow is an obvious place for representation of an alternate process if a fault is detected and a secondary communications channel (input or output) must be used. Any primary or secondary communications channels can be included in an OV-5 diagram.

Operational Activity Sequence and Timing Descriptions (OV-6)

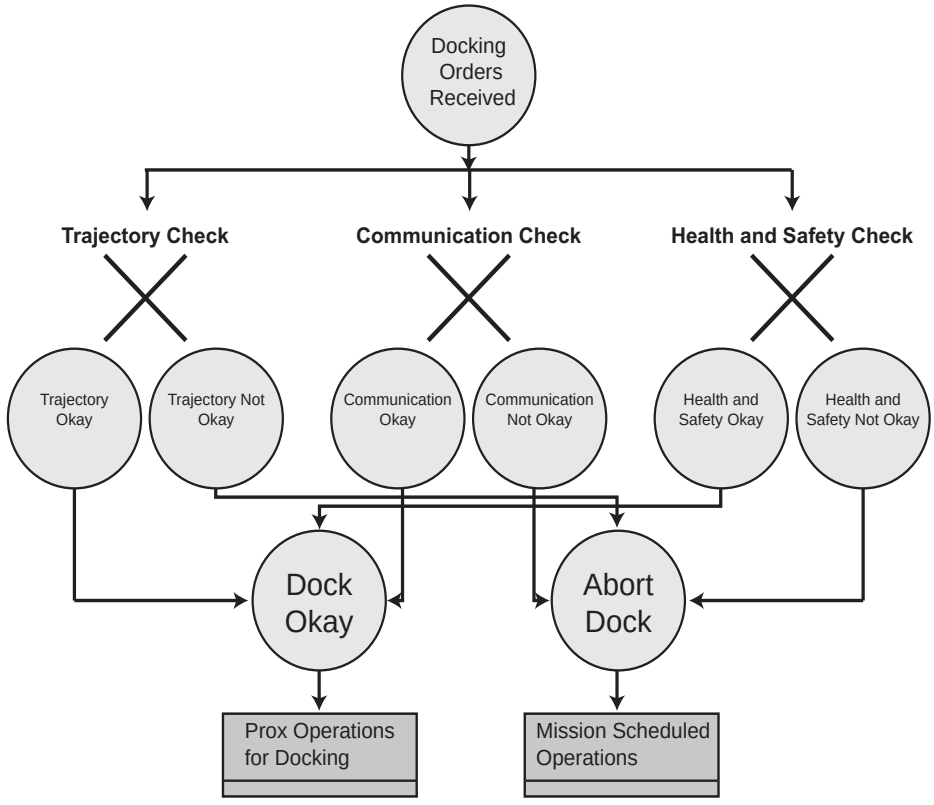
There are three types of OV-6 descriptions intended to show critical characteristics of dynamic sequencing and timing. Fault tolerance can be incorporated into each of them.

Operational Rules Model (OV-6a)

Figure 4 describes what a system does, using representations such as statecharts, Petri Nets, and process flow diagrams for the timing of processes and availability of operations. Statecharts are used in our example.

Fault tolerance, at least for known types of faults, can be represented here in detail. Icons listed as *trajectory not OK* and *communication not OK* represent a simple place to show the appropriate redundancy and fault isolation actions to ensure a fault-tolerant system operation.

Figure 4: *Operational Rules Model (OV-6a)*



Operational State Transition Diagram (OV-6b)

Figure 5 describes a system and its transitions from an object-centered view. A top-level OV-6b diagram is essentially a decision tree. Fault tolerance can be indicated in the form of messages when faults are detected. However, there seems to be little DoDAF support for tolerance of unanticipated faults (for which messages indicating faults may not be sent).

Operational Event-Trace Description (OV-6c)

Figure 6 (see page 26) can be represented as a set of timelines and interactions for each important physical object in the system with control capabilities. Fault tolerance, at least for known types of faults that can occur only within certain time constraints, can be represented here in detail.

Notice that in an OV-6c diagram, there are two kinds of events: those entirely on a single system and those that involve communication between systems. No assumptions can be made about what happens on a system's timeline until an external event linking the system to another occurs. Thus it is possible for the first, asynchronous, communication from the CEV to the ground system to be received *after* the second communication (the third arrow) from the CEV to the ISS is received by the ISS, regardless of it being sent from the CEV *before* the first asynchronous communication to the ground system. It is possible that an outside event can reach a system while the system is executing a wrapper, causing a fault and leaving the system in an inconsistent state. Examples of this are well known in operating systems; see for example [14]. Synchronous communications are represented using explicit acknowledgement.

The complete set of DoDAF *system views* is discussed next. These views describe systems and interconnections providing for, or supporting, organizational functions including both operational and business, as well as associate system resources to the OV. DoDAF system views are often required as part of the software development process, especially when developing complex systems. Particular emphasis is placed on those views where fault tolerance can be included most easily. No diagrams are provided for reasons of space; see the DoDAF Deskbook [5] for a more detailed description of these views.

Systems Interface Description (SV-1)

This identifies nodes, systems, and system items and their interconnections, mapping systems by their interfaces to the nodes and needlines described in OV-2. Any fault tolerance using backup systems and recovery routines is carried over directly from OV-2 diagrams. Systems responsible for fault tolerance, including backup systems can be specified here. However, no formal language has been provided for this specification.

Since there is no formal language or methodology to support fault tolerance directly within DoDAF, one workaround is to annotate these and all related diagrams. Unfortunately, the annotations will not be inserted directly into a CASE tool's information repository; hence, the CASE tool's internal consistency checking will not be available.

Systems Communications Description (SV-2)

This describes details of links in an SV-1 diagram into communications nodes, paths, and networks. Techniques to handle known classes of faults of specific nodes paths, or networks, can be incorporated into this diagram. Any hardware redundancy for fault tolerance, including standby systems, can be specified here.

Systems-Systems Matrix (SV-3)

This matrix shows relationships among systems in a given architecture such as system-type interfaces, planned versus existing interfaces, etc. In theory, it is useful in identifying alternate forms of communication between systems if there is a fault. However, there is no formal way of specifying such alternate communication within DoDAF. One potential workaround is to manually include the appropriate alternative relationships. As before, such annotations are not generally included within the CASE tool's internal representation of the systems within the architecture and, thus, the tool's consistency checks are not done automatically.

Systems Functionality Description (SV-4)

This describes functions performed by systems and the system data flows between system functions. Lower-level diagrams can be either functional or data-flow decompositions of top-level diagrams. Since faults theoretically can occur whenever data is created, processed, or stored, fault tolerance procedures can be incorporated directly into these diagrams and, therefore, into the CASE tool's internal repository for system consistency analysis.

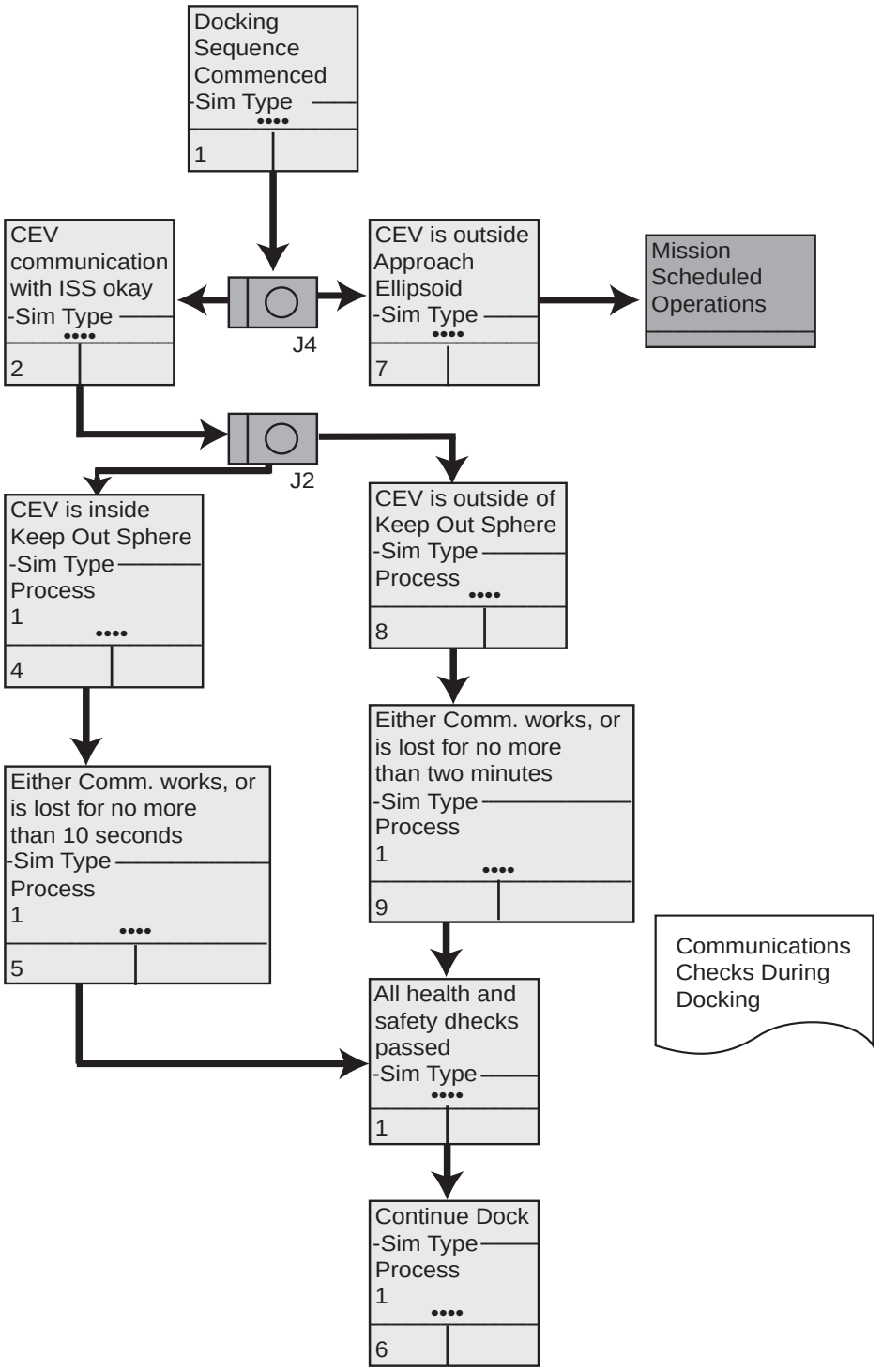


Figure 5: *Operational State Transition Diagram (OV-6b)*

Operational Activity to Systems Function Traceability Matrix (SV-5)

This provides mappings either of systems to capabilities or of system functions to operational activities. It is important to know which systems may be affected by a fault in an operational node; hence, fault tolerance is important here. Alternate safe states, recovery blocks, and redundant hardware can be specified for anticipated faults. Unfortunately, there is no obvious way to indicate alternative software/hardware paths in the event of an unknown failure.

Systems Data Exchange Matrix (SV-6)

This provides details of system data elements being exchanged between systems and their attributes. Since faults theoretically can occur whenever data is created, processed, or stored, fault tolerance can be incorporated anywhere in this matrix (as was the case with SV-4 diagrams).

Systems Performance Parameters Matrix (SV-7)

This provides performance characteristics

of SV elements for appropriate time frames. System faults due to distributed denial-of-service attacks can be addressed here. Delay-tolerant networks can serve as a useful recovery system. (An introduction to delay-tolerant networks is provided in [15].)

Systems Evolution Description (SV-8)

This lists planned incremental steps toward migrating a suite of systems to a more efficient suite or toward evolving a current system to a future implementation. This is the primary place where assessment of COTS products and their interfaces come into play when considering future fault tolerance.

Systems Technology Forecast (SV-9)

This is a text-based prediction of emerging technologies and software/hardware products, including assessment of COTS products and their interfaces, that are expected to be available in a given time period and that will affect future development of the architecture.

Systems Rules Model (SV-10a)

This identifies constraints that are imposed on system functionality due to some aspect of systems design or implementation. Such rules are often written in a relatively formal version of English (often called *Structured English*) using IF-THEN rules to indicate, say, timing or performance requirements. A more formal description than this would be useful for improving fault tolerance.

Systems State Transition Description (SV-10b)

This identifies responses of a system to events. The diagram is a systems-level version of OV-6b diagrams. This is clearly a place for fault tolerance, especially since linkages and nodes are placed directly into the CASE tool's internal repository for system consistency analysis.

Systems Event Trace Description (SV-10c)

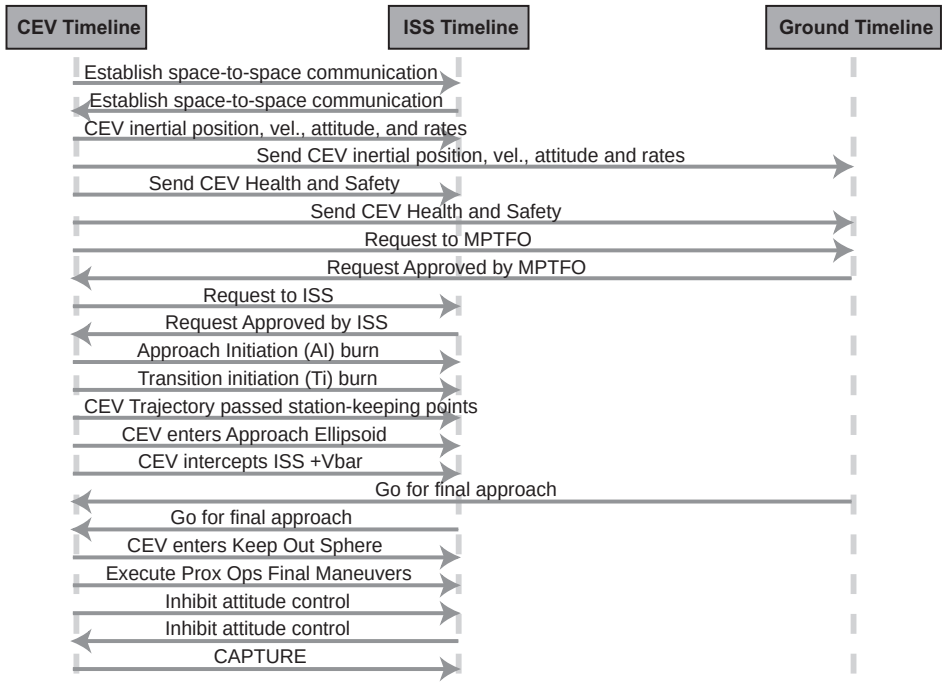
This is also used to describe system functionality. It identifies system-specific refinements of critical sequences of events described in OVs. The diagram is a systems-level version of OV-6c diagrams. This is clearly a place for fault tolerance especially since linkages and nodes are placed directly into the CASE tool's internal repository for system consistency analysis.

Physical Schema (SV-11)

This describes the physical implementation of the Logical Data Model entities to physical schema. Since this is the primary place where the failure of a server or data store is described in DoDAF views, hardware redundancy should be used, at least, here to improve system fault-tolerance.

There are two other types of DoDAF views that are not discussed here because they do not directly concern fault-tolerant systems: TVs, which list current and forecast standards, and *all views*, which include any overarching aspects, including doctrine; tactics, techniques, and procedures; goals and vision statements; concepts of operations; scenarios; and environmental conditions.

Figure 6: *Operational Event Trace Description (OV-6c)*



MPTFO: Mission Planning Training and Flight Operation

Conclusion and Further Work

It is clear that DoDAF is a complex framework that was intended for an even more complex problem – creating an SoS. The process for designing even a portion of a moderate sized system using DoDAF and a typical CASE tool is time-consuming, requiring several online tutorials and multiple presentations, along with hands-on help. One would expect strong support for fault tolerance, especially since DoDAF was intended initially for military systems. Indeed, the timing requirements that could be represented easily, say, in the OV-6a, OV-6b, and OV-6c operational views, and SV-10a, SV-10b, and SV-10c system views might include fault tolerance.

However, the reality is otherwise. The three most readily available sources of DoDAF documentation – the DoDAF deskbook, and volumes I and II of the DoDAF architectural framework – do not even mention the word fault. This is a strong indication of the lack of support for fault tolerance in the use of DoDAF within a systems engineering process. Again, this suggests an opportunity for having fault tolerance research incorporated into large-scale systems engineering methodologies and into high-quality industrial CASE tools.

There are many opportunities for the dependability community to increase research collaborations, especially in the engineering of SoS. The development of specific graphical notations for fault tolerance, such as coordinated atomic actions, and their eventual incorporation into commercial CASE tools and large architectural frameworks provides the potential to have fault tolerance built into systems rather than being an add-on.

Development of an expert system module to be incorporated into CASE tools holds considerable promise. Fault tolerance is critical, and support for it is likely to become more important in the future. Perhaps this support can be in the form of an expert system add-on to advise on fault tolerance, or in creation of a graphical notation that explicitly supports fault-tolerant designs and implementations. Having such support early in the life cycle can have a great effect on the creation of real fault-tolerant systems.

The convergence of the computer security and dependability fields is highly encouraging, since it provides an opportunity to study fault isolation and mitigation techniques during denial-of-service and

similar attacks.

A major goal of this article is to provide an impetus for the dependability community to affect the functionality of CASE tools and frameworks to provide early life-cycle support for fault tolerance. Clearly, the lack of early life-cycle support for fault tolerance makes it more difficult to include it within the design of complex SoS. It remains to be determined if there will be enough perceived return on investment in order for CASE tool vendors to add fault tolerance to DoDAF.◆

Acknowledgement

This research was partially supported by the National Science Foundation under grant number 0324818. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either express or implied, of the U. S. Government.

References

1. Randell, B. "System Structure for Software Fault Tolerance." IEEE Trans. Software Engineering 11.2 (1975).
2. Avizienis, A., and J.P. Kell. "Fault Tolerance by Design Diversity: Concepts and Experiments." Computer Aug. 1994: 67-80.
3. CMMI Product Team. "CMMI for Development." Version 1.2. Technical Report CMU/SEI-2006-TR-008. Pittsburgh: Software Engineering Institute (SEI), Carnegie Mellon University (CMU), 2006.
4. Capability Maturity Model. Version 1.0. Pittsburgh: SEI/CMU, 1991.
5. DoD. DoDAF v1 Deskbook. DoD Architecture Working Group, 2004 <https://acc.dau.mil/CommunityBrowser.aspx?id=31667>.
6. DoD. DoDAF v1 Volume I. DoD Architecture Working Group, 2004 <jitc.fhu.disa.mil/jitc_dri/pdfs/dodaf_v1v1.pdf>.
7. DoD. DoDAF v1 Volume II. DoD Architecture Working Group, 2004 <jitc.fhu.disa.mil/jitc_dri/pdfs/dodaf_v1v2.pdf>.
8. Mittal, Saurabh. "Extending DoDAF to Allow Integrated DEVS-based Modeling and Simulation." JDMS: The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology 3.2 (2006).
9. Voas, Jeffrey M. "COTS and High Assurance: An Oxymoron?" Proc. of the 4th IEEE International Symposium on High-Assurance Systems Engineering (HASE '99), November 17-19 1999, Washington: IEEE Computer Society, 1999.
10. Voas, Jeffrey M., and Jeffrey E. Payne. "Dependability Certification of Software Components." Journal of Systems and Software 52.2-3 (2000): 165-172.
11. Corman, David. "The IULS Approach to Software Wrapper Technology for Upgrading Legacy Systems." CrossTalk Dec. 2001 <www.stsc.hill.af.mil./crosstalk/2001/1201>.
12. Leach, Ronald J. "Can this COTS-Based System Be Saved?" PC/104 Embedded Solutions 9.4 (2005): 38-44.
13. Oliveira Guimaraes, Jose. "The Green Language Exception System." The Computer Journal 47.6 (2004): 651-661.
14. Silberschatz, A., Galvin, P., and G. Gagne. Operating Systems Concepts. New York: John Wiley, 2005.
15. Li, Jiang, et al. "Customizable Localized Computation of Connected Dominating Sets for Self-Organizing Wireless Networks." Proc. of the Second International Conference on Embedded Software and Systems (ICESS'05). IEEE, 2005.

About the Author

Ronald J. Leach, Ph.D., is professor and chair of the Department of Systems and Computer Science at Howard University. He does research in software engineering with special interest in reuse, metrics, fault tolerance, performance modeling, process improvement, and the efficient development of complex software systems. Leach is the author of five books and more than 65 other published technical articles. He has bachelor's, master's, and doctorate degrees in mathematics from the University of Maryland, and a master's degree in computer science from Johns Hopkins University. He has three children, a terrific grandson, two grandcats, and one granddog.

Department of Systems and Computer Science
School of Engineering
Howard University
Washington, D.C. 20059
Phone: (202) 806-6650
Fax: (202) 806-4531
E-mail: rjl@scs.howard.edu