IEEE SCAN-LIKE INTERFACE FOR AIR TRAFFIC CONTROL SOFTWARE TESTING

Thérèse M. Smith, BAE Systems and Air Traffic Software Architecture, Inc. (USA and Canada)
Carol E. Uri, Federal Aviation Administration, Washington, D.C.

Abstract

The benefits of Modified Condition/Decision Coverage (MC/DC) in development of safety critical software are well recognized.

Use of commercial-off-the-shelf (COTS) software is encouraged, and vendors of COTS software have expressed interest in developing a standard interface to assist in carrying out functions needed for the safety critical domain.

A useful analogy can be drawn between a complex integrated circuit, with millions more storage locations and logic gates inside than pins on its interface, and an executable version of software, with many more variables and logical statements inside than are directly visible on its interface description.

IEEE has developed an interface standard for testing package integrated circuits (that is, the tests are conducted through the interface pins), called Boundary-Scan. An analogous interface specification for software modules has been proposed in a COTS software forum, initially by Thérèse Smith and continued by Dr. Gregory Eakman of Pathfinder Solutions, who has completed scholarly work in this area.

The FAA is advancing the technology used to support interface metadata. Net-centric architecture increases both the visibility of the metadata repository and the level of automation used to interact with the metadata. Given that the software vendors are pursuing a strategy for testability, there is interest at FAA in whether the testability is useful, and if so, in expanding the metadata to be maintained in a proposed “FAA Interface Management System” to include interface data used for testing.

Introduction

Problem to be Solved

Operating costs are high now, and increased safety and functionality are desired.

Maintenance of Air traffic Control (ATC) systems software is a significant cost, and it has been suggested that some cost savings can be achieved by use of commercial off-the-shelf (COTS) products. In addition to the Information Technology Management Reform Act (or Clinger-Cohen Act) of 1996, which among other things encourages the acquisition of commercial information technology products [1], a number of National Airspace System (NAS) Architecture programs mention movement toward COTS as a goal. These include Aeronautical Data Link (ADL), Air Traffic Management (ATM), NAS Infrastructure Management System (NIMS), Operational and Supportability Implementation System (OASIS), Airport Surface Detection Equipment – Model X (ASDE-X), Flight Data Input and Output (FDIO) COTS Replacement Phase II, and others [2].

When considering changing an implementation, it is necessary to maintain safety, and desirable to improve safety.

There are multiple approaches to safety of software, and testing is a significant one. Because the commitment to safety is strong, ATC software must be thoroughly assured, and the current practice implies that it must be thoroughly tested.

The definition of “thorough” depends upon the potential of the software to cause safety-related failures identified in a system safety assessment. If there is more potential for the software to cause safety-related failures, the acceptable degree of thoroughness is more demanding.

Systems may have to be tested more than once in their lifetime. The environment in which
software is to run can change. For instance, platform upgrades are likely to occur, and software components sharing the platform can be expected to change. This implies that the "tested" status of a software component can change. Software must therefore be "testable" so that a change in the software's environment can be followed by testing.

The benefits of Modified Condition/Decision Coverage (MCDC) in development and testing of safety critical software are well recognized [3].

Constraints on the Solution

While COTS software has undoubtedly been tested to some degree, it might be the case that the initial target market for COTS software might not have included market segments requiring the same degree of thoroughness of testing appropriate to the ATC environment in which its use is contemplated.

For example, vendors of flight systems have expressed an interest in using object-oriented software, and in some flight systems, the appropriate degree of testing is Modified Condition/Decision Coverage (MCDC). MCDC applies to software whose failure would cause or contribute to a catastrophic failure of an aircraft [4].

While a software purchaser might be willing to supplement the testing that the vendor has carried out, it can also be the case that the intellectual property of the vendor would become more generally known if, for example, source code were shared in support of additional testing. For the same reason, COTS vendors might not want to produce or release such thorough documentation of their software to enable white box testing of their product.

The desire for thoroughly tested and testable COTS whose intellectual property features are not revealed might be met by an adaptation of a testing method standardized by the Institute of Electrical and Electronics Engineers (IEEE) called Boundary-Scan [5], used in the testing of integrated circuits.

Approach

Some software vendors interested in expanding their markets to include safety critical uses have participated in an activity under the aegis of the Object Management Group (OMG).

The analogy between logic implemented in digital hardware and logic implemented in software occurred to author Therese Smith while reading [6], [7], and [8]. Within the OMG's Safety Critical, Transportation and Testing working group activities, Ms. Smith suggested that the IEEE Boundary-Scan interface would provide to the vendors, by analogy, an approach to constructing testable software without necessarily revealing the design decisions about the internals of the software. A draft Request for Proposal (RFP) was produced [9]. This suggestion has been pursued over several years by Dock Allen of MITRE Corporation, who leads the OMG Safety Critical activity, and by Dr. Eakman of Pathfinder Solutions, who has recently submitted a proposal [10] apropos of the current version [11] of the abovementioned RFP. This line of work continues Dr. Eakman's scholarly work [12].

Given that the software vendors are pursuing a strategy for testability, there is interest at the FAA in determining whether the testability is useful, and if so, in expanding the metadata ("facts about data") to be maintained in an envisioned repository called an "Interface Management System" to include interface data used for testing. Registration of interface metadata, i.e., descriptive characteristics of an interface, is discussed later in this paper.

It will be helpful to briefly describe some related ideas. Those ideas include combinatoric vs. sequential logic, an analogy between software and hardware implementations of logic, and a description of a test method used in hardware. This test method could be applied to software design, such that the resulting software is testable without having its structure revealed.

Combinatoric vs. Sequential Logic

It is useful to distinguish between those logic operations that involve memory and those that do not; for example, the result of 4+5 does not depend upon memory, but the result of 5 more than what was previously accumulated does require memory.

Logic whose implementation does not use memory is combinatoric logic. It can be tested by applying inputs, observing outputs, and comparing outputs to those expected from the operation of the logic itself upon the inputs.
Sequential logic makes use of memory. In order to know whether an implementation of such logic is correct, a sequence of inputs must be provided, and the final output observed for comparison with the outputs expected for correct operation for that sequence of outputs. For example, if we are testing an accumulator, then in order to see whether it correctly accumulates 5 we must not only provide the input 5, but also we must establish the content of the previously accumulated amount to assure ourselves that 5 can be correctly added to whatever the previous amount might be.

Establishing the content of a memory within an implementation of logic is not generally a simple process. The memory can be remote from the inputs. The result is that testing can be elaborate.

**Analogy between Software and Hardware Implementations of Logic**

Because the distinction between combinatoric and sequential is present within the notion of logic, it continues to manifest itself in both the hardware and the software implementations of logic.

Memory appears as latches or registers in hardware. In software, these latches or registers are often referred to with variable names.

As a demonstration of the analogy, one example of each of a hardware and a software implementation of the following logic shall be given:

There is a cockpit display of nearby traffic, and the location at which the symbol for an aircraft is to be displayed depends upon certain factors. If there is a resolution advisory condition between that aircraft and the ship containing the display, and if the display is in multifunction display mode, then the track position as determined by the Traffic Alert and Collision Avoidance System (TCAS) is the track position at which the symbol for that aircraft is to be displayed. In all other cases, the Automatic Dependent Surveillance - Broadcast (ADS-B) track gives the position of the symbol for that track.

An implementation of this logic in hardware is shown in Figure 1, Hardware Implementation.

Figure 1. Hardware Implementation

An implementation of the same logic in software is shown in Figure 2, Software Implementation.

Figure 2. Software Implementation

In both cases, the implementations make use of memory, namely a memory for the track position developed by the TCAS algorithm and the position developed by the ADS-B algorithm.

In order for a test of this logic to be carried out, the content of the memory must be known, and preferably the content of the memory should be set to various desired values.

**Utility of Separating Combinatoric and Sequential Logic in Testing**

The benefit of separating the logic into memory and combinatoric logic is that we can test...
memory in a fashion suited to memory, and we can test combinatoric logic in a fashion suited to it.

With memory, we need to test whether input patterns are remembered such that they are properly provided as outputs when appropriate. With combinatoric logic, we can test by providing inputs and checking outputs.

If there are many inputs, the number of tests grows large. MC/DC [13] provides a technique to reduce the number of tests to those needed.

Those steps in testing whose purpose is to establish the content of memory can be reduced to a minimum if a test interface provides the capability to set the memory contents.

**Analogy between Circuit Environment and Software Environment**

In the context of testing integrated circuits, having a correct design is not sufficient to ensure that a fabricated circuit will work. The material of which the circuit is fabricated is a manufactured product of great complexity. Defects can occur in the circuit materials that render that instance of implementation of a circuit incapable of performing properly.

The desire to test at the instance level makes sense for integrated circuits.

It is customary to differentiate software errors from hardware errors as systematic errors, i.e., errors that are “always there” and manifest themselves when the conditions allow. Of course, we can imagine systematic errors in hardware as well. It would seem that the most significant difference is that hardware always has its material of which it is fabricated, and possible imperfections in this material warrant testing at the instance level.

It might be useful, though, to consider that software can also have dependencies upon its environment. Software can behave differently on different platforms, with different environment variables, though such dependencies could be considered to be undesirable.

If we consider a circuit to be the combination of the circuit design, the fabrication process as directed by the design, and the environment in which the fabricated part is operated, and if we consider a software instance to be the combination of a software design, compilation and installation and the software and hardware environment in which the assembled module is operated, we might conclude that testing at the instance level is warranted.

When instance level testing is warranted, it is helpful to have a method of setting memories, checking their content including by the normal output path, and establishing inputs to combinatoric logic sections and observing the output of combinatoric logic sections.

In the case of integrated circuits, it is desirable to accomplish this function after the instance is packaged, i.e., though interface pins, a very small number of them compared to the number of memories to be checked.

In the case of software instances, it is desirable to accomplish this function in a way that does not necessarily reveal the software implementation, i.e., vendors can sell testable software without giving away their design.

These two constraints appear to be analogous because the package can be seen as hiding the implementation: The test pattern going in and the test pattern expected to emerge from the boundary might not be sufficient to reveal the design specifics.

**Description of Test Method in Hardware**

The IEEE Boundary-Scan interface [5-8] allows a test pattern, as a serial stream of bits to be fed into an integrated circuit from its boundary, and a result, as a serial stream of bits to be received from the integrated circuit boundary. Bits from this serial stream are used as inputs to memory cells and inputs to combinatoric circuits. Outputs from memories and from combinatoric circuits are directed to the output test stream.

**Application of Test Method to Software**

If a software interface permitted the setting of memories (variables) and the setting of inputs to combinational logic and permitted reading of memories and obtaining of output of logic combinations, and if the values to be set were assigned a place in a serial stream, and the values
output were assigned a place in a serial stream, then vendors could specify that certain segments of the serial input stream corresponded to certain segments of the serial output stream, and how changes on the input test pattern should result in changes in the output stream.

MC/DC can guide the selection of test patterns to be generated. Thus DO-178B level A testing [4] can be applied through this interface. By using fewer test patterns, less demanding testing can also be accommodated, though designer, developer and tester judgment would guide the reduction of test vectors.

**Interface Metadata**

This test pattern and test result and relationship data becomes part of the metadata of a software application. Because the test pattern is applied at a test interface and sampled at a test interface, it is interface metadata.

FAA has an evolving capability for supporting metadata. Part of this capability consists of an existing application called the FAA Data Registry, plus accompanying metadata creation, registration, and configuration management processes for standardizing application-independent data exchange specifications. This application is envisioned to expand into a full-fledged Interface Management System (IMS). The following sections describe FAA's current metadata registration activities and the potential for including testing-related metadata among the metadata to be managed in the IMS.

**Current Metadata Registration Activities and the FAA Data Registry**

National Airspace System modernization requires the ability to share information widely among NAS users and service providers and across domains. NAS architects foresaw as early as 1997 that transitioning to a reengineered information architecture that treats operational data as NAS-wide assets and not as items defined by, and embedded in, individual programs would require careful planning and a lot of coordination. It was then that the NAS Information Architecture Committee, or NIAC, was chartered by the FAA's NAS Configuration Control Board to proactively address NAS-wide information access and interoperability issues.

A major part of NIAC's work is to oversee the creation and registration of standardized data exchange specifications for commonly shared NAS data, with the expectation of meeting the following objectives:

- Enhance information system interoperability by reducing the requirements to translate and transform data
- Reduce cost and time to develop, implement, and maintain systems
- Provide uniform descriptions and representations of commonly shared data
- Improve data integrity and accuracy
- Manage interface requirements down to the data element level.

A data standard is a recognized and accepted description of a unit of data that includes a name, a precise definition, a prescribed structure, and other descriptive characteristics, or metadata (facts about data). Since the NAS itself is a "system of systems," most of which were designed to operate within domains and not across them, the same information is often described differently by each system and the process of harmonizing these descriptions can be very labor-intensive. This is accomplished via a process developed by NIAC in 2001 and described at the 20th DASC Conference [14] in which subject matter experts prepare case files of proposed data standards that will eventually go to the NAS CCB for final approval as NAS-level requirements.

Once a case file is approved by the NAS CCB, the data standards are maintained in an Oracle-based FAA Data Registry [15] in accordance with FAA's Data Management Policy (Order 1375.1C) [16] and FAA-STD-060b, Data Standard for the NAS [17]. There they are available for incorporation in interface requirements documents. FAA-STD-025e, Preparation of Interface Documentation [18] provides developers with guidance on how to refer to applicable

---

2 This name has been proposed for the metadata repository/registry function associated with the FAA's System-Wide Information Management (SWIM) effort [21].
standards or, if none exist that would suit their particular requirement, how to provide the missing metadata for subsequent documentation in the FDR.

**FAA Data Registry**

The FAA Data Registry is a tool for recording, publishing, and maintaining metadata about application-independent data exchange standards. It provides information about the precise meaning of data, and it provides a place to capture information during the development of data standards. FDR is based on the ISO/IEC 11179 standard (ISO = International Organization for Standardization, IEC = International Electrotechnical Commission) entitled Metadata Registries [19]. The purpose of ISO/IEC 11179 is to support the identification, definition, registration, classification, management, standardization, and interchange of data elements and to promote the sharing and exchange of data throughout the international community.

In a 11179-compliant registry like FDR, metadata is registered in appropriate contexts, i.e., the application environment or scope in which the metadata has meaning. Some of the contexts may be restricted to the casual browser. FDR is available on the Internet at http://fdr.faa.gov, and guests are allowed to read and search for registered items with FAA-wide applicability (i.e., within the FAA context). Users must have individual accounts and passwords to read or edit items registered in other contexts. Figure 3 presents a screen shot of a typical search for en route metadata.

![Figure 3. Typical FDR Search](image)

An example of an approved data standard is shown in Figure 4, in this case the observed cloud layer height as recorded and represented in Aviation Weather Report (METAR) format.

![Figure 4. Typical Data Standard](image)

At present, metadata stored in the FDR is limited to what is required to fully describe, in an application-independent fashion, the individual elements of data shared across interfaces among NAS systems and subsystems. The following is a partial list of the kind of metadata maintained in the FDR that specifies a data element:

- Preferred name
- Definition
- Context and context definition
- Data ID and version number
- Alternate names, including name type, context, and language
- Classification scheme associations
- Effective begin/end dates
- Associated Data Concept name and definition
- Associated Value Domain name and definition
- High/low value range
- Unit of measure and precision
- Data type
- Maximum/minimum length
- Interchange format
- Character set
- Permissible values and value meanings
- Associated Conceptual Domain name and definition

10.C.5-6
Examples of actual data instances
Additional descriptive comments
Associated documents, including document name, type, language, URL, description
Related data elements and relationships, e.g., an element may be compound, an object class with methods, calculated from one or more other elements, etc.
Steward organization (the organization responsible for maintaining this data element specification)
Registration status – incomplete, certified, standardized, retired, etc.

In future, the metadata attributes and the tool that manages them will be expanded to capture the physical and functional characteristics of the interfaces in which the data elements are exchanged, including the source and sink of each interface. This new Interface Management System (IMS) will play a major role in transitioning FAA to a net-centric environment similar to the environment envisioned by the Department of Defense [20].

**SWIM and the Interface Management System**

A key strategy for achieving net-centricity in the NAS will be carried out through a concept called System-Wide Information Management, or SWIM [21]. In SWIM, interfaces among portions of ATC automation are documented in metadata for each element of information exchanged, including the source and sink of each interface. The IMS will subsume the capabilities of the FDR and expand the focus from documenting application-independent data exchange specifications to facilitating the search and discovery of actual data as well as metadata. The expanded set of metadata will need to include such things as:

- Precision and accuracy levels
- Security Classification (sometimes called Quality of Protection)
- Quality of Service (QoS) metrics, including latency in delivery for data handled by a network.

To conduct interface software testing as described in this paper, the FAA will need to consider defining additional appropriate measures and descriptive metadata, e.g., level of testing undergone, signals used for testing, etc.

**Conclusion**

Commercial software is advancing rapidly in testability. At the same time, FAA is advancing rapidly in metadata technology. Recognition that testing artifacts are interface metadata allows combination of these two fronts of advancement.

One result is that testability is less of an obstacle to the use of COTS software than it has been.

Testability of integrated circuits has been the subject of much thought, and is analogous to a type of testing of software that accommodates MC/DC testing.

There might be additional benefits to be obtained by attempting to transfer technology of testing from the hardware domain across to the software domain.

FAA’s ability to manage interface metadata (store, search for, discover and retrieve) enables the testability technology to be applied to software component applications of the ATC system.

Testability technology supplied by IEEE 1149.1 includes the ability to construct systems from subsystems, and to construct the corresponding test inputs for the system from the test inputs of the subsystems [3]. Composed systems can be tested with the same structured approach. This implies that the choice exists to discontinue the practice of developing test environments specific to the version of the subsystem. A further choice exists concerning the placement of the position of the domain of expertise for testing among developers, system integrators and FAA. If the position of testing is mainly within FAA, the opportunity to share the test facility over the several development programs exists. The effect upon testing of the schedule for platform upgrades is focused on the effect, if any, upon test vectors. Cost savings appear possible; however, analysis would be required to determine whether a net savings would be expected.
Acknowledgements

The authors would like to thank Mr. Jay Merkle of FAA’s En Route Systems Integrated Product Team for supporting the initial work at Object Management Group; Professor John Stankovic of the University of Virginia for introducing them to the work of Professor David Lorge Parnas (McMaster University, Hamilton, Ontario) on software errors [22], [23], [24]; and FAA’s Operations Planning System Engineering Directorate for continuing support.

References


Disclaimer
The contents of this material reflect the views of the author(s). Neither the Federal Aviation Administration nor the Department of Transportation makes any warranty or guarantee, or promise, expressed or implied, concerning the content or accuracy of the views expressed herein.

Email Addresses
Thérèse M. Smith, therese_smith@alum.mit.edu
Carol Uri, carol.uri@faa.gov

10.C.5-9