The Object-Oriented Design of Intelligent Test Systems

John W. Sheppard and Gerald C. Hadfield

Abstract

Recent IEEE and Department of Defense (DoD) initiatives to standardize automatic test equipment (ATE) architectures are providing unique opportunities to improve the development of test systems. The purpose of A Broad Based Environment for Test (ABBET) initiative is to provide the next generation in ATE and test program set (TPS) development through the standardization of test services and test development tools. The Artificial Intelligence and Expert System Tie to Automatic Test Equipment (AI-ESTATE) initiative will provide architectures for standardizing the test services of reasoning systems and provide guidance for test strategy services supporting the ABBET architecture. In this paper we focus on one of the standards proposed under AI-ESTATE for developing diagnostic models that proposes models for the object-oriented design of fault trees and enhanced diagnostic inference models. We will also discuss the applicability of the AI-ESTATE P1232.1 Data and Knowledge Specification standard to object-oriented testing, and we will propose an architecture for combining ABBET and AI-ESTATE into a comprehensive ATE system.

Introduction

Recent initiatives by the IEEE and the DoD that will standardize automatic test equipment architectures provide a unique opportunity to improve the development of test systems. The ABBET initiative (IEEE PAR 1226) is attempting to provide the next generation in ATE and TPS development by standardizing test services and test development tools. By using Ada in developing test executives, test programs, and communications protocols, ABBET-compliant test systems will be interoperable, incorporate transportable software, and move beyond vendor product specific test stations. In addition, the AI-ESTATE initiative (IEEE PAR 1232) will provide architectures for standardizing the test services of reasoning systems (including traditional fault trees, expert systems, model-based systems, simulation-based systems, and neural networks) and provide guidance for test strategy services supporting the ABBET architecture.

Central to the ABBET and AI-ESTATE initiatives is the desire to incorporate the object-oriented paradigm in specifying test resources, test programs, and test services. The various component standards of ABBET and AI-ESTATE are using information modeling based on the International Organization for Standardization (ISO) Express object-oriented modeling language. In this paper, we focus on one of the standards proposed under AI-ESTATE for developing diagnostic models: AI-ESTATE P1232.1 Data and Knowledge Specification. This particular standard proposes models for the object-oriented design of fault trees and enhanced diagnostic inference models. In addition, the proposed standard includes specifications for attribute models and common element models. These correspond to specific classes of objects that would be specified for a particular test architecture.

In this paper we discuss applicability of the AI-ESTATE P1232.1 standard to object-oriented testing, and we specifically discuss the motivation for a new standard for automatic testing and the history of ATE equipment in the military and the commercial sector. In addition, we address object-oriented design for reusing test resources (both software and hardware), integrating heterogeneous reasoning systems (for intelligent ATE), and increasing the flexibility of testing that arises from using encapsulated testing. We describe a proposed architecture for combining the ABBET and AI-ESTATE standards and conclude by describing the advantages of applying a standard ATE architecture and object-oriented ATE/TPS design arising from the ABBET and AI-ESTATE initiatives.

Object-Oriented Design

Object-oriented design (OOD) is quickly becoming a popular approach to developing complex systems [1]. Typically, OOD is limited to the development of complex software, but the paradigm can be applied to complete

system design and modeling. It is this latter approach that is behind recent efforts in the IEEE to develop ATE-related standards. OOD relies on specifying structure and behavior of system components in order to maximize component reuse and organization. The object-oriented paradigm arises from a mathematical concept called the abstract data type and adds a mechanism for inheriting characteristics through a class hierarchy (sometimes referred to as a semantic network [2]).

Abstract Data Types

An *abstract data type*
consists of a specification of a set of objects together with the operations capable of being performed on these objects [10]. The end result of abstracting data types in this way is that they become implementation independent. Conventional data types define the structure of data. Abstract data types, on the other hand, define the logical behavior of values of data. This in turn specifies an interface between users of the type, e.g., designers, programmers, and end users, and the implementation of the type. Only the specification is required to use the type, and only the specification is referred to when implementing the type.

For an ATE system, we can consider the various elements of the tester to be abstract data types. For example, for the instruments of the tester, the objects are defined by the instruments, and the behaviors, i.e., the functions operating on the data, are the instructions available on the instrument. The abstract data type corresponding to a voltmeter consists of a specification of the meter and a collection of functions such as "set scale" or "return voltage."

In specifying an abstract data type, a designer must identify syntactic information (called a *signature*) and semantic information. The signature of the abstract data type consists of a list of names of the functions regarded as the basic operations of the type. The semantic information consists of specifications of the behavior of these operations. Semantics of data types can be expressed axiomatically, operationally, or denotationally. Implementing the data types consists of piecing together the objects and providing the operations (through software or hardware) specified.

Inheritance

Specifying the abstract data types is the first step in applying object-oriented analysis and design. OOD attempts to organize the data types into classes of related types. In this way, information common to more than one abstract data type can be abstracted further from the basic objects and operations. This results in a lattice of types and a lattice of classes. But for such abstraction to occur, a mechanism is required for members of a class to retain the attributes of that class without explicitly storing these attributes with the members. The mechanism provided is called *inheritance*.

Object-oriented systems apply either single inheritance or multiple inheritance, depending on the complexity of the system being designed. With single inheritance, a simple hierarchy of classes is constructed and all members of a particular class receive the attributes of that class without those attributes being explicitly specified for the member object. For example, suppose we have a class of objects called power_supplies. Associated with this class, we may have defined several specific objects corresponding to individual power supplies, and these individual specifications would include information relevant to the particular power supply (such as bus_address or power_level). But several characteristics common to all power supplies may be specified at the class level, i.e., power_supplies would have specific attributes such as power_range or constant_rms_level, and the members of the class would have the attributes of the class as well as their individual attributes. These general attributes are specified only once.

When an object can receive general attributes from more than one "parent" in the hierarchy, then a mechanism for *multiple inheritance*
must be provided. Multiple inheritance also abstracts attributes from the individual objects to a class level, but since conflicts can arise, a means of conflict resolution must also be provided. In addition, default values are frequently provided for attributes that have the effect of overriding inherited values under certain circumstances.

Express Modeling Language

Although not generally considered an object-oriented design problem, data modeling can benefit from the principles of OOD. Data modeling is generally not considered an application area for OOD because methods need not be defined. Nevertheless, the concepts of data encapsulation and inheritance that are provided by OOD have tremendous

applicability to data modeling.

As a means for modeling data using the object-oriented paradigm, the ISO developed a standard data modeling language called Express, which is an implementation-independent modeling language. Express is ideal for OOD in that it does not specify how the data of a particular model will be stored in a computer or how the data will be accessed or manipulated. To allow the exchange of models, some physical structure must be provided as a standard exchange format. ISO STEP 10303.21 provides a file format for data exchange. The standards being produced by the IEEE Standards Coordinating Committee 20 (SCC20) for ATE are using Express as one of their modeling languages.

ATE Architectures

Automatic test equipment has been vastly improved since the first appearance of sequential state, paper-tape-driven "automatic" test systems of 20 years ago. The advent of high-speed digital computers, microprocessors, sophisticated instrumentation, and bus structures has resulted in state-of-the-art automated test systems capable of isolating failures in equipment operating at digital rates exceeding 150 MHz [4]. Unfortunately, the advance of software used to harness and drive this powerful instrumentation has failed to keep pace. Repair facilities still struggle with test programs that require "full-up" instrumentation systems and can take little or no guidance from skilled technicians. These are the same problems that plagued the operators using the paper-tape-driven systems. Such problems suggest a need to overhaul the current approach to controlling test equipment by incorporating standard architectures for ATE systems and incorporating intelligent approaches to fault isolation.

Classic Architectures

A typical ATE system includes a suite of test instruments in a common test station, a test control computer to control the instruments and interpret results of tests, an interface test adapter that connects the unit under test (UUT) to the test station, a test program set that instructs the computer on how to test the UUT, and a test executive that determines the order in which tests are run and instructs the computer on how to drive the instruments [8]. Until now, test programs and test executives have been developed to test a UUT by following predetermined fault trees. TPS developers have attempted to minimize the mean time to fault isolate by constructing more efficient fault trees. This approach has worked to a point, but the problem with writing TPSs around any fixed fault tree is the lack of flexibility in the resulting system. Separate fault trees and TPSs are often required for different symptoms, different optimization criteria, and different instrument suites. Further, should an instrument fail, typically, the fixed fault tree will fail to reach a conclusion.

Military Architectures

Current ATE systems are cumbersome and subject to errors in the way they perform diagnosis. Precomputed fault trees are used to guide the test program, and sometimes false conclusions are reached. Further, ATE systems are incapable of adapting to failed instruments or insights from the technician.

Several initiatives are in progress to standardize ATE architectures. However, these efforts have focused on ways to standardize and integrate analysis tools and instrument interfaces. Two current military architectures are the Consolidated Automated Support System (CASS) and the Integrated Family of Test Equipment (IFTE).

CASS was designed with the intent of reducing the lifecycle cost of systems and ensuring that ATE systems are standardized to meet that need. CASS provides an industry definition of operational constraints and maintenance policy for automatic test. Further, the test systems were to be available to weapon system designers so that issues of testability could be addressed early, and more time could be spent focusing on system performance [9].

IFTE attempted to meet the need for increasing tactical flexibility. IFTE was to provide ATE equipment for line replaceable units (LRUs) close to their operational units and was to replace existing test methodologies for all of the maintenance levels in the Army. The end result is a shorter logistics chain for system support [5].

Commercial Architecture

As with the military, there are several commercial initiatives in progress to incorporate automatic testing techniques.

The commercial airline industry has gone the furthest in standardizing on ATE equipment, and initiatives in the automotive industry appear to be making progress.

The T-100 system [6] is a fully integrated diagnostic system developed by EDS for General Motors. Its purpose is to provide a means for automatically testing automobiles in the maintenance shop by connecting umbilical lines directly to the automobile. The system evaluates the car and produces a fault tree for the mechanic to follow.

The Standard Modular Avionic Repair and Test (SMARTTM) system is the standard for avionics ATE for the airline community. It utilizes ARINC Specifications 608A, 626, and 627 to provide a standardized test environment that allows freedom of choice in selection of test instruments and test control computers [8]. SMART is currently being used by several avionics manufacturers and third party ATE vendors to develop automatic test systems to support avionics for new generations aircraft including the A330/A340, B-777, and MD-11. At present, three airlines are using SMART to maintain avionics from their operational fleet: Lufthansa German Airlines, Air France, and Delta Air Lines.

Standards for ATE

Recently, considerable effort has been taken by the IEEE to develop standards for ATE. The standardization efforts are occurring under the administration of the IEEE SCC20. The ABBET and AI-ESTATE standards are particularly relevant to the object-oriented design of ATE as discussed in the following sections.

A Broad-Based Environment for Test--ABBET

The initial motivation behind developing ATLAS was to provide unambiguous communication of test requirements. ATLAS eventually evolved into a test language capable of performing functions of a specification language and a programming language.

The Ada programming language was developed to support advanced software engineering practices and to foster the development of reusable code for large software systems. Though not strictly an object-oriented language, Ada provides many of the capabilities required by OOD.

In the late 1980s, initiatives were in progress to incorporate Ada into test systems while preserving the high-level specification capabilities of ATLAS. To accomplish this, the IEEE established Project Authorization Request 1226 for the ATLAS/Ada-Based Environment for Test (ABET). Since that time, ABET has evolved considerably, and today it exists as ABBET.

ABBET is a set of software interface standards for the test domain that enables exchange of test information and the development of object-oriented test equipment. These interface standards are defined to support software portability, reusability, exchangeability, and interoperability and to serve as targets for test-related software development tools.

The ABBET architecture consists of a framework of standardized service interfaces and information representations. The primary services are underlying procedures used by ABBET application programs that perform specific functions for the end users. Specifically, three sets of services are defined: test and diagnostic services, information services, and user interface services. The test and diagnostic services provide the interface between the test programs and the test resources. They also provide services between the UUT and the test system. The information services enable access to both internal and external test-related information. Finally, the user interface services provide the interface between a human user and the test system.

Artificial Intelligence and Expert System Tie to ATE-- AI-ESTATE

The AI-ESTATE standard P1232 is being developed to standardize the interfaces between test systems and AI-based systems. In addition, AI-ESTATE is including standard representations for several types of knowledge bases and databases. Currently, the standard specifies representations for fault tree models (FTMs) and enhanced diagnostic inference models (EDIMs).

AI-ESTATE is being developed using a cooperative processing model. Under this model, all processes communicate across a communication pathway or bus and access other parts of the system through a set of services. Thus, each

functional block of an AI-ESTATE system consists of an object in the sense of OOD.

Specifically, each functional block is defined by the operations that can be performed on or by that block. The attributes of the block are specified in a class lattice in order to maximize reuse of components. If one functional block needs to interact with another, it uses the services provided by that block.

Using this architectural concept, the objects communicating in an AI-ESTATE system include the test system, the reasoner, the presentation system, a maintenance data collection using a database management system, the UUT, and the operating system. This architectural concept is depicted in Figure 1. Any data used by the objects on the communication pathway will be specified in some standard representation. Even though we currently have specifications for only the FTM and the EDIM, we anticipate defining or referencing standards for maintenance data collection databases and other databases and knowledge bases.

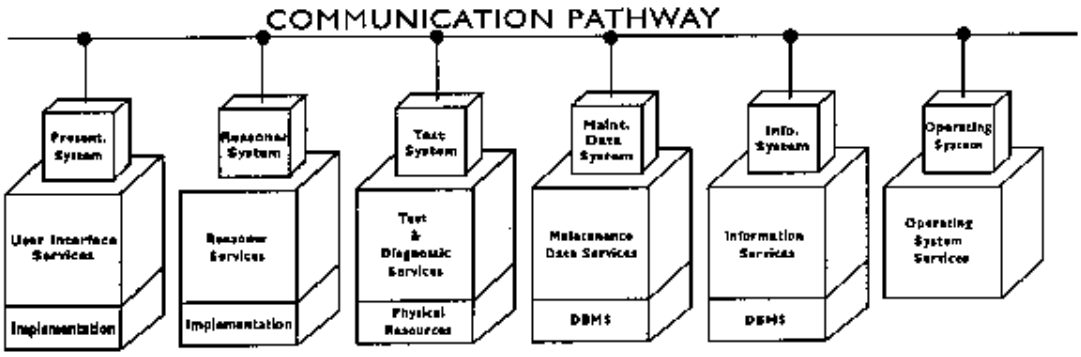

FIGURE 1: ARCHITECTURE FOR AN AI-ESTATE TEST SYSTEM.

Knowledge Representation

The current model representations defined by the AI-ESTATE P1231.1 Data and Knowledge Specification are specified using the Express modeling language. Currently, AI-ESTATE has specified four models including the Common Element Model (a collection of entities used by all models), the Attribute Model (a collection of attributes inherited by several of the models), the Fault Tree Model (a standard representation for a static fault tree), and the Enhanced Diagnostic Inference Model (a standard representation that extends the concepts of the dependency model and the information flow model).

Fault Tree Models

The structure of a fault tree can be viewed as a decision tree or table. Each node of the tree corresponds to a test with some set of outcomes. The outcomes of the tests are branches extending from the test node to other tests or fault isolation conclusions. Typically, TPSs are designed around static fault trees; therefore, the AI-ESTATE subcommittee developers decided it was prudent to include a representation for the fault tree in its standard even though fault trees are not typically considered AI systems. Fault trees serve as the minimum representation for test-related data.

The AI-ESTATE fault tree model inherits elements and attributes of the common element model and the attribute model. The fault tree is processed by starting at the first test step, executing the indicated test, and traversing the branch corresponding to the test outcome. The procedure is followed recursively until a leaf is reached in the tree, indicating that fault isolation has occurred.

Enhanced Diagnostic Inference Models

The EDIM is based on concepts of dependency modeling [4] and information flow modeling [11]. As with the FTM, the EDIM inherits elements and attributes from the common element model and the attribute model. In EDIMs, diagnostic test outcomes are generalized beyond pass and fail outcomes to multiple outcomes for a diagnostic test. Inferences are identified between particular test outcomes and other test outcomes and diagnostic conclusions.

The set of inferences associated with a particular test outcome are represented in disjunctive normal form. This representation provides flexibility and consistency in the logical expression of the inferences. No negations are permitted within the AI-ESTATE representation because negative terms are expressly specified as atoms in the conjunctions. In other words, each conjunction consists of a set of positive and negative inference atoms, and the negative inference atoms are assumed to be negated.

EDIMs can be used by an AI system in several ways. The most basic approach is to limit test outcomes to pass and fail and to assume the test outcomes are symmetric. This leads to a traditional dependency model as used by weapon system testability analysis and system testability analysis tool. By allowing multiple failure inference, various grouping operations, asymmetric inference, and reasoning under uncertainty, the model extends to the information flow model as described in [11].

Finally, the EDIM extends further by permitting more detailed specification of test and UUT data, multiple test outcomes, and a generalized attribute. This generalized attribute provides tremendous expressive power. For example, using the generalized attribute, causal relationships between fault isolation conclusions can be specified, thus extending the EDIM to include a causal model. The generalized attribute can also be used to include constraint information when diagnosing by constraint satisfaction. Thus, EDIMs contain considerable power for providing many AI-based solutions.

Test Encapsulation

An important notion in developing object-oriented test systems that include AI for test sequencing and inference is the ability to isolate a test from the remainder of the test set. This type of test isolation is referred to as *test encapsulation*
[7]. An encapsulated test is an atomic test element defined to be independent of the current state of the UUT and the tester. To define an encapsulated test, preconditions and post-conditions of the test need to be defined. The preconditions indicate the steps necessary to set up the test for execution, and the post-conditions indicate the steps necessary to return the state to neutral. Individual tests and groups of tests can be encapsulated depending on the specific requirements of the tester and the UUT.

Combining ABBET and AI-ESTATE

To produce an object-oriented test system, it has become apparent that both ABBET and AI-ESTATE offer capabilities that contribute to solving the problem. As such, the two standards need to be combined so that the combination maximizes the capabilities of both. The ABBET committee envisions that AI-ESTATE will provide the test strategy services for the test system with ABBET in control. The AI-ESTATE committee, on the other hand, envisions that ABBET will provide test services with control being distributed throughout the objects of the test environment. Thus, each object interacts concurrently with any other object on the bus (within the limitations of the bus), and control is provided by the object requesting a service at a particular point in time.

The architecture of a combined ABBET and AI-ESTATE test system is shown in Figure 2. As in Figure 1, the architecture consists of six objects, but in this architecture the services are distributed between the two standards. ABBET would provide services for the presentation system (the user interface services), the information system (the information services), and the test system (the test and diagnostic services). Within the test system, we see the reliance on the ABBET architecture to use virtual resources and access to encapsulated test procedures. We could consider the UUT to be a separate object on the communication pathway, but we will assume that it is accessed through the test system for this architecture. AI-ESTATE, on the other hand, will be responsible for providing services to the reasoner and the maintenance data collection system. Further, AI-ESTATE will define the formats of the databases and knowledge bases used in the test system. Finally, standard operating system services will be provided by whatever operating system is used on the ATE. Figure 2 shows UNIX being selected as the operating system.

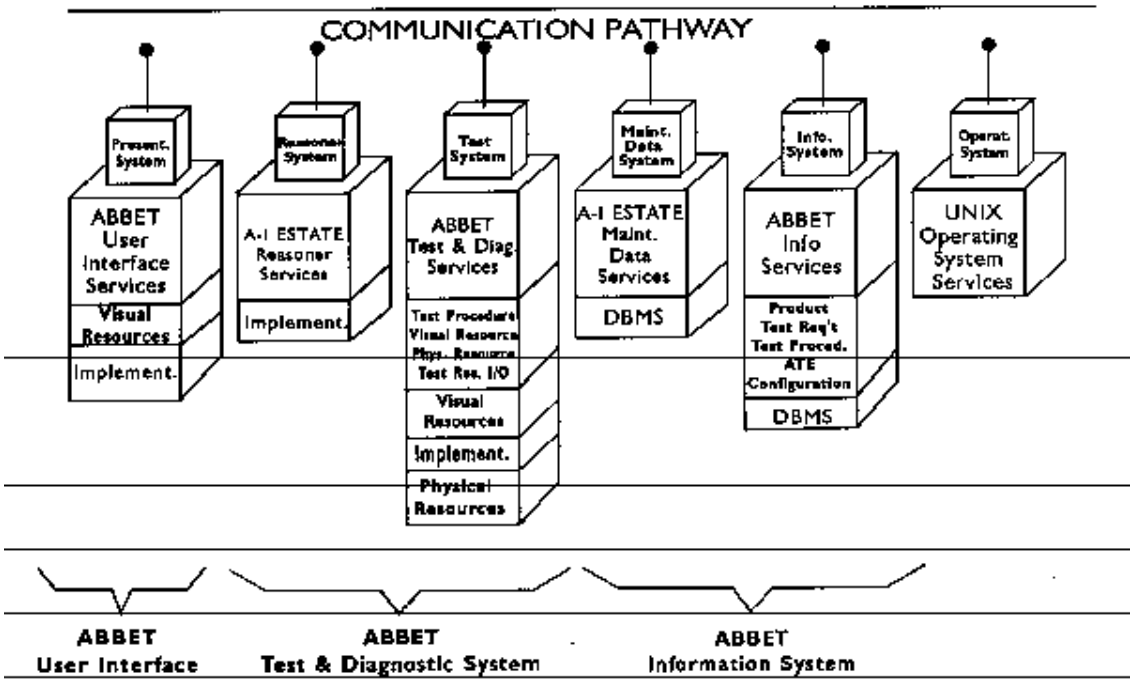

FIGURE 2: COMBINING ABBET AND AI-ESTATE.

Advantages of a Standard ATE System

Several advantages can be realized in applying a standardized object- oriented design of an ATE system.

- The object-oriented paradigm facilitates the reuse of test resources. These resources include test programs, instruments, drivers, and other objects within the test system.
- The object-oriented paradigm for test encapsulation provides an easy mechanism for adding and deleting test objects as required.
- A standard form of knowledge representation provides a means of sharing knowledge between homogeneous and heterogeneous reasoning systems. As a result, maintaining the knowledge in the knowledge base is simplified.
- The object-oriented paradigm incorporating the virtual resource concept enables the test to be configured to required levels of performance. This results in a decrease in development cost and maintenance cost.

Several additional advantages exist for using the standards and the standardization process in determining an architecture for intelligent ATE. These include the availability of a large pool of expertise in refining the architecture, the development of a consistent, well-thought-out product, the acceptance by a large group of people, and the use of a formalized review process to ensure the quality of the product. By not using the standards, the cost of incorporating these features in a project not employing standards would be prohibitive.

Conclusions

Using the object-oriented paradigm guides the development of two ATE-related IEEE standards: ABBET and AI-ESTATE. As discussed, OOD is quite effective in designing a complex system such as an intelligent ATE, and several benefits can be realized. As we look to the remainder of this century and into the next, we recognize the need for advanced techniques to continue to evolve ATE systems and architectures. This evolution must include the use of AI because system complexity continues to grow unabated. To date, no standards have been defined for incorporating AI in test systems. With the ABBET and AI-ESTATE standards, we have the opportunity to shape the future of automatic testing and to provide a mechanism for grappling with the growing complexity.

John W. Sheppard
ARINC Research Corporation
2551 Riva Road

Annapolis, MD 21401-7465
Voice: 410-266-2099
Fax: 410-266-4010
Internet: sheppard@arinc.com

Gerald C. Hadfield
ARINC Research Corporation
1925 Aerotech Drive, Suite 212
Colorado Springs, CO 80916-4219
Voice: 719-574-9001
Fax: 719-574-2594
Internet: ghadfiel@arinc.com

References

1. Booch, Grady, *Object Oriented Design*, Benjamin Cummings, 1991.
2. Brachman, Ronald J., "On the Epistemological Status of Semantic Networks," in *Associative Networks: Representation and Use of Knowledge by Computers*, N.V. Findler ed., New York: Academic Press, 1979, pp. 3-50.
3. DePaul, R. Jr., "Logic Modeling as a Tool for Testability," *AUTOTESTCON '85 Symposium Proceedings*, 1985 IEEE Automatic Test Conference, Uniondale, New York, September 1985, pp. 203-207.
4. Dill, Harry H., "Test Program Sets - A New Approach," *AUTOTESTCON '90 Conference Record*, 1990 Automatic Test Conference, San Antonio, Texas, September 1990, pp. 63-69.
5. Espisito, C.M., G.A. Walz, R. Burchacki, and R.J. Carnevale, "U.S. Army IFTE Technical and Management Overview," *AUTOTESTCON '86 Symposium Proceedings*, 1986 IEEE Automatic Test Conference, San Antonio, Texas, September 1986, pp. 319-322.
6. GM, "GM Service Information and Diagnostic Technology," *General Motors Techline*, 1989.
7. Haynes, L., W. Simpson, S. Goodall, J. Sheppard, and F. Phillips, "Test Strategy Component of an Open Architecture for Electronics Design and Support Tools," *AUTOTESTCON '92 Conference Record*, 1992 IEEE Automatic Test Conference, Dayton, Ohio, September 1992, pp. 49-56.
8. Melendez, E.M., and D.C. Hart, "Airlines Get SMART for Avionics Testing," *AUTOTESTCON '90 Conference Record*, 1990 IEEE Automatic Test Conference, San Antonio, Texas, September 1990, pp. 505-508.
9. Najaran, M.T., "CASS Revisited - A Case for Supportability," *AUTOTESTCON '86 Symposium Proceedings*, 1986 IEEE Automatic Test Conference, San Antonio, Texas, September 1986, pp. 323-327.
10. Reade, Chris, *Elements of Functional Programming*, 10 Wokingham, England: Addison-Wesley Publishing Company, 1989.
11. Sheppard, John W., and William R. Simpson, "A Mathematical Model for Integrated Diagnostics," *IEEE Design and Test of Computers*, Vol. 8, No. 4, December 1991, pp. 25-38.
