IEEE P1671 ™/D2
Draft Trial-Use Standard for Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML

Prepared by the Test Information Integration (TII) Subcommittee of the

IEEE Standards Coordinating Committee 20 on Test and Diagnosis for Electronic Systems

IEEE Standards Activities Department
Standards Licensing and Contracts
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331, USA

Abstract: This document specifies the framework for the family of ATML standards. ATML defines a standard exchange medium for sharing information between components of an automatic test system (ATS), utilizing the extensible markup language (XML).
Keywords: Automatic Test Markup Language (ATML), XML Schema, ATML Instance Document, Automatic Test Equipment (ATE), Automatic Test System (ATS).

Introduction

(This introduction is not part of IEEE P1671 /D2, Draft Trial-Use Standard for Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML.)

The benefits of using standards in test-related applications have long been recognized. The scope for standardization extends from low-level standards associated with test instrument control to high-level standards associated with specifying tests in an implementation-independent manner.
In the 1960s, Aeronautical Radio, Incorporated (ARINC) started the development of the Abbreviated Test Language for Avionics Systems (ATLAS). In 1976, management of the ATLAS standard was passed to the IEEE, and the ATLAS acronym was changed to Abbreviated Test Language for All Systems to reflect its broader field of applications.

Within the IEEE, development of ATLAS and ATLAS-related standards was vested in an ad hoc committee, which later became the IEEE Standards Coordinating Committee 20 (SCC20). In the mid-1980s, SCC20 broadened the scope of its activities to embrace other standards projects related to test and diagnosis, including Automatic Test Program Generation (ATPG), Test Equipment Description Language (TEDL), Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE), A Broad Based Environment for Test (ABBET), Software Interface to Maintenance Information and Collection Analysis (SIMICA), Receiver Fixture Interface (RFI), Signal and Test Definition (STD), and the Automatic Test Markup Language (ATML).

This standard provides the framework for a family of standards providing specifications for test-related applications and environments. This family incorporates object-oriented technology and information modeling to specify language-independent elements within a wide variety of test environments, including Built-In Test (BIT) systems, Automatic Test Systems (ATS), and manual test environments. Each of these interfaces is specified in the form of a XML Schema.

XML Schemas define the basic information required within any test application and provides a vehicle for formally defining the test environment by defining a class hierarchy corresponding to these basic information entities and provides several methods within each to enable basic operations to be performed on these entities. ATML component standards within the ATML framework define the particular requirements within the test environment.

Patents

Attention is called to the possibility that implementation of this trial-use standard may require use of subject matter covered by patent rights. By publication of this trial-use standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents or patent applications for which a license may be required to implement an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Participants

At the time this draft trial-use standard was completed, the Test Information Integration (TII) Subcommittee had the following membership:

Chris Gorringe, *Co-Chair*

Mike Seavey, *Co-Chair*

Nasir Ahmed
Steve Allen
Peter Allum
Tony Alwart
Michael Araiza
Hagay Azar
Charles Barker
Keith Beard
Malcolm Brown
Antonius Bunson
Giampiero Casalegno
Kevin Coggins
Bernard Dathy
Tim Davis
David Droste
James Dumser
Tamara Einspanjer
Crocket Ellis Jr.
Keith Ellis
Leo Errico
Oscar Fandino
Robert Fox
William Frank
Thomas Gaudette
Scott Gearhart
George Geathers
Anthony Geneva
William Gerstein
Arnold Greenspan
Kyle Gupton
Arthur Hann
Susan Harbour
Michelle Harris
Michael Harrison
Robert Hayes
Hans Hopf
Ashley Hulme
David Hunter
Ivor Isaacs
Anand Jain
Gary Jones
Patrick Kalgren
Mark Kaufman
Bernhard Kausler
Michael Keller
John Knowles
Arthur Larsen
Teresa Lopes
Tracy McQuillen
David Mills
Scott Misha
Mukund Modi
Ion Neag
Travoris Nunn
Les Orlidge
Steve Osella
Cedric Pendelton
Henry Perry III
Duy-Huan Pham
Dan Pleasant
Dave Ptacek
Narayanan Ramachandran
Rabindranath Raul
Peter Richardson
Lou Roberts
Robert Robinson
David Rohacek
Paul Salopek
Howard Savage
John Sheppard
Joe Stanco
John Stratton
Walter Struppler
David Thomas
Sylvain Tourangeau
Jeorg Urban
Tim Wilmering

The following members of the balloting committee voted on this trial-use standard. Balloters may have voted for approval, disapproval, or abstention.

(to be supplied by IEEE)

CONTENTS

Draft Trial-Use Standard for Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML

1. Overview

The family of Automatic Test Markup Language (ATML) standards is being developed under the guidance of the Test Information Integration (TII) subcommittee of the IEEE Standards Coordinating Committee 20 (SCC20) to serve as standards for product test. The ATML family of standards specifies a comprehensive environment for integrating design data, test strategies and requirements, test procedures, test results management, and test system implementations. The family of ATML standards includes reference to IEEE Std. 1232 (AI-ESTATE), IEEE P1636 (SIMICA) and IEEE Std. 1641 (STD). These referenced IEEE standards are therefore part of the ATML family.

1.1 Scope

ATML defines a standard exchange medium for sharing information between components of automatic test systems. This information includes test data, resource data, diagnostic data, and historic data. The exchange medium is defined using the extensible markup language (XML). This document specifies the framework for the family of ATML standards.

1.2 Purpose

The purpose of ATML is to support test program, test asset, and Unit Under Test (UUT) interoperability within an automatic test environment. ATML accomplishes this through a standard medium for exchanging UUT, test and diagnostic information between components of the test system. The purpose of this document is to provide an overview of ATML goals as well as to provide guidance for usage of the ATML family of standards.

1.3 Application

This trial-use standard provides an overview of the ATML family of standards for developing the following:

— ATML-conformant systems.

— Design data for use in test.

— ATML environment for tools.

— Shared usage of maintenance data and the results of testing.

Anticipated users of the ATML family of standards include the following:

— Unit Under Test (UUT) developers.

— UUT maintainers.

— Test Program Set (TPS) developers.

— TPS maintainers.

— Automatic Test Equipment (ATE) system developers.

— ATE system maintainers.

— Test instrument developers.

— Developers of ATML-based tools and systems.

— Developers of prime mission equipment that use the supported UUT as a component.

1.4 Conventions used within this document

The sub-clauses present an overview (background and purpose) of ATML, how ATML applies to the lifecycle of a product, and defines the ATML framework.

This trial-use standard uses the vocabulary and definitions of relevant IEEE standards. In case of conflict of definitions, except for those portions quoted from standards, the following precedence shall be observed: (1) Clause 3; (2) SCC20 documentation and standards; and (3) *The Authoritative Dictionary of IEEE Standards, Seventh Edition*.

For clarity, portions of IEEE Std. 1232, IEEE Std. 1641, and IEEE P1636 have been repeated within this trial-use standard. In the event of revision to IEEE Std. 1232, IEEE Std. 1641, or IEEE P1636, the current, approved version of that IEEE Standard takes precedence.

2. Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

— IEEE Std. 1232-2002, IEEE Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE).

— IEEE P1636, IEEE Standard for Software Interface to Maintenance Information and Collection Analysis (SIMICA).

— IEEE Std. 1641-2004, IEEE Standard for Signal and Test Definition (STD).

3. Definitions

For the purposes of this draft trial-use standard, the following terms and definitions apply. *The Authoritative Dictionary of IEEE Standards, Seventh Edition*, should be referenced for terms not defined in this clause. In the event a term is explicitly redefined, or further defined in an ATML component standard, the component standards definition shall be normative for that ATML component standard.

3.1 Abbreviated Test Language for All Systems (ATLAS): A standard abbreviated English language used in the preparation (and documentation) of test procedures or test programs. The test procedures or test programs are implemented either manually or with automatic or semiautomatic test equipment.

3.2 adapter: A device or series of devices designed to provide a compatible connection between the test subject and the test equipment. *Synonyms*: interface device; interface test adapter; test adapter.

3.3 anomaly: Deviation from the normal behavior of a test subject. Faults (e.g.; output stuck high, gain low) and manufacturing defects (e,g,; missing or incorrect components, incorrectly installed components) are kinds of anomalies.

3.4 Application: (A) The use to which a system is put. **(B)** The use of capabilities provided for by a system specific to the satisfaction of a set of users requirements.

3.5 ATML instance document: *See*: instance document.

3.6 Automatic Test Equipment (ATE): Equipment that is designed to conduct analysis of functional or static parameters to evaluate the degree of performance degradation and that may be designed to perform fault isolation of unit malfunctions.

3.7 Automatic Test System (ATS): Includes the Automatic Test Equipment (ATE) as well as all support equipment, software, Test Program (TP), and adapters.

3.8 behavior: A formal representation of the characteristics that describe the operation, function, relationships, control, or static properties of a test entity.

3.9 class: A template for the creation of an object instance. The class defines the properties of an object.

3.10 classification: A grouping of objects on the basis of common characteristics.

3.11 context: Reflects the intended scope of a set of tests. Examples of context include manufacturing process test, maintenance test, design verification test, screening test, etc.

3.12 corrective action: Intended to eliminate anomalies. Corrective actions include repair, replacement, calibration, alignment, and other services. *See also*: maintenance.

3.13 diagnosis: The conclusion(s) resulting from tasks, tests, observations, or other information.

3.14 diagnostic controller: The agent (this could be from an expert/reasoner system or from an operator) that invokes test procedures in the sequence required to achieve test goals.

3.15 diagnostic data: That information which supports the investigation and analysis of the cause or nature of a condition, situation, or problem through all phases of a system life cycle.

3.16 diagnostic knowledge: Provides the information required to support the diagnostic process. This knowledge defines the relationships between possible test outcomes and anomalies that may cause these outcomes.

3.17 diagnostic process: A structured combination of tasks, tests, observations, and other information used to localize a fault or faults.

3.18 diagnostic procedures: *See*: diagnostic process.

3.19 element: A bounded component of the logical structure of an XML document that has a type and that may have XML attributes and content [adapted from XML 1.0 Recommendation].

3.20 encapsulation: The grouping of data, and operations upon that data, into a single object.

3.21 entity: A distinct thing, object, or concept. (The Artificial Intelligence Dictionary, Ellen Thro, MICROTREND Books, San Marco, CA.)

3.22 error logging: The recording of an error condition detected during the execution of a service.

3.23 fault: A degradation in performance due to detuning, mis-adjustment, mis-alignment, or failure of part(s).

3.24 framework: A collection of classes created specifically to serve the needs of an application area.

3.25 functional parameters: Any specific quantity or value affecting or describing the measureable characteristics of a unit being considered which behaves as an independent variable or which depends on some functional interaction of other quantities. (Derived from MIL-STD-1309D)

3.26 functional partitioning: The logical separation of system or unit elements along interfaces that define and isolate these elements on the basis of function or purpose.

3.27 functional test: A test that is intended to verify that a test subject is behaving as specified.

3.28 global attribute: An attribute declaration that is a child of the xs:schema element. A global attribute can be applied to any element.

3.29 Go/NoGo Test: Terms referring to the condition or state of operability of a unit that can only have two outcomes, GO, functioning properly, or NO-GO, not functioning properly. (Adapted from MIL-STD-1309D)

3.30 handler: A program or routine that performs or controls one task. (e.g.; error detection)

3.31 historical data: All relevant information available concerning the product, tests, and test equipment. This includes test observations (raw measurement data) derived test outcomes (i.e.; LO, HI, GO), diagnostic conclusions derived from performing tests and the knowledge base, test subject mission and configuration history, test resources mission and history, etc.

3.32 information modeling: An information model is a formal description of types of ideas, facts, and processes, which together form a model of a portion of interest of the real world and which provides an explicit set of interpretation rules. *See also*: corrective action. (Information modeling the EXPRESS way, D. Schenck and P. Wilson, New York: Oxford University Press, 1994)

3.33 instance document: An information set, grouped for some purpose, that is governed by a single XML Schema.

3.34 instrument: A device whose purpose is usually the generation or measurement of a class of signal.

3.35 integrated diagnostics: A structured process that maximizes the effectiveness of diagnostics by integrating testability, automatic and manual testing, training and technical information to provide a cost effective capability to detect and isolate faults with the ultimate goal of maximizing equipment availability and minimizing total user cost.

3.36 interface: A shared boundary that specifies the interconnection between two units or systems, hardware or software. In hardware, the specification includes the type, quantity, and function of the interconnection circuits and the type and form of signals to be interchanged via those circuits. In software, the specification includes the object type and, where necessary, the name or instance handle of specific objects copied or shared between the two systems.

3.37 Interface Definition Language (IDL): A machine-compatible language used to describe interfaces that clients call and implementations provide. IDL provides a neutral way to define an interface. [IDL is an Object Management Group (OMG) product.]

3.38 Interface Device (ID): *See*: adapter.

3.39 Interface Test Adapter (ITA): *See*: adapter.

3.40 knowledge-based test: A test based in part on previously acquired information. (Adapted from MIL-STD-1309D)

3.41 language-independent specification: The format for describing services that is not tied to any specific computer language.

3.42 maintenance: Activity intended to keep equipment (hardware) or programs (software) in satisfactory working condition, including replacements, adjustments, repairs, software/firmware updates, and program improvements. Maintenance can be preventative or corrective. (Adapted from MIL-STD-1309D)

3.43 manual testing: Testing that requires a human to execute some or all of a test procedure.

3.44 manufacturing defect: (A) A product anomaly. **(B)** Any non-conformance with the specified requirements of the product.

3.45 mapping: Process of correspondence between the elements of one set and the elements of another set.

3.46 markup declarations: XML element type, XML attribute-list, XML entity and XML notation declarations that provide a grammar for a class of XML documents.

3.47 method: A property of a class that defines a specific behavior.

3.48 object: A member (instance) of a class that encapsulates the data (state) and the behavior (methods) of the object. (The Artificial Intelligence Dictionary, Ellen Thro, MICROTREND Books, San Marco, CA.)

3.49 object instance: An specific occurrence of an object.

3.50 observation: The raw data acquired by executing a test procedure. It represents the observed characteristics of a specific signal (e.g.; the voltage peak of a sinusoid wave form), the observed characteristics of the environment (e.g.; the ambient temperature), or the derived value of product characteristics (e.g.; the measured value of gain).

3.51 open architecture: An architecture from which a system can be assembled from multiple vendor-supplied interface components. The resulting system can execute applications written by arbitrary independent vendors and can be extended by users other than the original supplier.

3.52 operation: An action defined by a procedure.

3.53 parametric testing: Testing of a test subject's ability to function correctly within acceptable tolerance when all values are varied within specified limits. (Adapted from MIL-STD-1309D)

3.54 portability: The ease with which software can be transferred from one system or environment to another. A relative measure of effort, inversely proportional to the level of modification required for software to be transferred from one system or environment to another.

3.55 process: Sequence of operations performed in and by the equipment in which the variable is to be controlled.

3.56 product characteristic: An observable attribute of a product. This includes functional, physical, and performance characteristics (e.g.; gain and bandwidth of an amplifier).

3.57 resource manager: A process or activity that initializes and manages the resources in a system.

3.58 response data: The information sensed from a test subject as the result of an applied stimulus.

3.59 sense signal: The response taken or measured from a test subject.

3.60 sequencer: An object that controls the execution flow of programs.

3.61 service: Operation or run-time call whose behavior and interface are standardized. *See also*: method.

3.62 signal: (A) The behavior controlled or observed by a test resource. **(B)** A visual, audible, or other indication used to convey information.

3.63 software product: A complete set of computer programs, procedures, associated documentation and data designated for delivery to a user.

3.64 static parameters: Variables given a constant value for a specific purpose or process.

3.65 stimulus: Any physical or electrical input applied to a test subject intended to produce a measurable response. (Adapted from MIL-STD-1309D)

3.66 task: The smallest unit of work subject to management accountability. (e.g.; a sequence of instructions treated as a basic unit of work by an operating system)

3.67 test: (A) An observed activity that may be caused to occur (e.g.; stimulus-response) in order to obtain information about the behavior of a test subject. **(B)** A set of stimuli, either applied or known, combined with a set of observed responses and criteria for comparing these responses to a known standard. (Adapted from IEEE Std.1232-2002)

3.68 testability: A design characteristic that allows the status (operable, inoperable, or degraded) of an item to be determined and the isolation of faults within the item to be performed in a timely and efficient manner. (Adapted from MIL-STD-1309D)

3.69 test adapter: *See*: adapter.

3.70 test asset: An assemblage of instruments, interconnect devices, supporting software, and manual procedures that enable one or more test objectives to be achieved. *See also*: automatic test system.

3.71 test control: The functionality that directs and facilitates the execution of tests and the collection of data.

3.72 test entity: A specific procedure or action that will be taken to determine a test subjects capabilities or limitations.

3.73 test method: A specification that defines the algorithm, procedures, and required controllable inputs and potential behavior (nominal and anomalous) of a test subject.

3.74 test object: Any object defined for use within the domain of test representing an encapsulated view of a test method with interfaces to a test system.

3.75 test objective: The purpose of a specific procedure or action to be performed on a test subject.

3.76 test outcome: A mapping from an observation to one of a set of discrete possibilities.

3.77 test procedure: The implementation of a test method.

3.78 test program (TP): A program specifically intended for the testing of a test subject.

3.79 test program set (TPS): A assembly of items necessary to test a test subject on a piece of Automatic Test Equipment (ATE). This includes the electrical, mechanical, instructional, and logical decision elements. The individual elements of the TPS are the TP, the adapter, and the TPS documentation (TPSD).

3.80 test requirement: A specification of the test methods and test conditions needed to evaluate and diagnose a test subject.

3.81 test specification: A document that defines the tests to be performed on a test subject to verify conformance with its performance specification, without reference to any specific test equipment or test method.

3.82 test strategy: (A) The arrangement of specific tester types to achieve optimum throughput and diagnostic capability at the least possible cost given the fault spectrum, process yield, production rate, and product mix for a particular environment. (Adapted from MIL-STD-1309D) **(B)** A selection of test methods to achieve some diagnostic test within execution time and test resource constraints.

3.83 test subject: The specific product design that is the focus of attention or target for the development of tests and diagnostics.

3.84 user interface: The part of the application that permits the user and application to communicate with each other to perform certain tasks.

3.85 virtual instrument software architecture (VISA): The general name given to the VPP (VXI Plug&Play) 4 Specification and its associated architecture. The architecture consists of two main VISA components: the VISA resource manager and the VISA instrument control resources.

3.86 XML schema: The structure or framework used to define a data record. This includes each field’s name, type, shape, dimension, and mapping.

4. ATML Overview

4.1 Background

Market and technology pressures cause the need for improved testing environments. There are market pressures to reduce a new product's time-to-market and to reduce maintenance costs. At the same time, testing environments must respond to continuously evolving and increasingly complex technology used in modern products. Test technology needs to keep pace with these pressures.

The lack of accepted industry standards for test related information is one factor that has limited the availability and use of improved test environments, Computer-Aided Design (CAD) environments and exchange standards for design data have helped designers to cope with these pressures, but testing environments have lagged behind.

Computer-aided test environments offer the potential to efficiently access and manipulate design and test information. For example, product test information from earlier life cycle phases can be preserved in order to eliminate rediscovery efforts in later phases. Libraries of reusable test procedures can be provided for various types of products. Test equipment interfaces, capabilities, and control characteristics can also be accessed from libraries and used to adapt existing tests to other environments with different test resources.

4.2 Purpose

The primary purpose of ATML is to specify standards for test environments that encompass the total product life cycle. ATML defines an integrated set of test related information that supports the information needs of test environments for testing applications. ATML is intended to accomplish the following objectives:

— Facilitate the communication, sharing, and reuse of product design and test information for the purpose of testing the product.

— Facilitate TPS portability and interoperability.

— Facilitate instrument interchangeability.

— Facilitate the development, integration, and use of test software and test software development tools.

— Support the application of integrated diagnostics.

— Supports modular software architectures based upon a framework that supports reusable software products.

4.3 ATML and the Product Life Cycle

There are primarily five reasons for performing tests on products:

— *Design Verification*. Verify that a product design meets its functional performance specifications.

— *Product Verification*. Verify that a product meets its functional and structural specifications.

— *Product Calibration*. Adjust the characteristics of a product to meet specified tolerances or performance criteria. Calibration tests are often incorporated as part of a verification process.

— *Product Maintenance*. Identification of a fault condition(s) in the event a product verification test fails or a problem is reported with the product.

— *Process Control*. Acquire information about a process used to produce, operate, or maintain a product. For example, monitor and maintain proper environmental conditions during a manufacturing process.

These reasons motivate the creation of test assets. Test assets can be manually or automatically controlled; extend from simple inspection to complex functional processing; be built in or applied by external means; or be applied on entire systems or constitute subassemblies. The decisions regarding the test assets depend on the selected life-cycle strategies. All tests produce potentially useful data that can be used for evaluation as dictated by either technology or policy. ATML supports the useful exchange of test information among product life-cycle phases while supporting the different test-related technologies present in each.

The product life cycle shown in the top portion of Figure 1 is typical of product development, deployment, operation, and maintenance operations. The life cycle phases are shown from left to right across the diagram. Activities performed during each phase are shown as circles on the diagram, with the test activities being highlighted. The three life cycle phases that have test activities are as follows:

— *Design*. The product proceeds from a concept to a working prototype or preproduction model. The characteristics of the model are tested to verify compliance with specified design and performance criteria.

— *Manufacture*. The product is fabricated and made available for distribution and use. Tests are used to assure that the product is properly assembled prior to delivery.

— *Support*: The product is performing in its intended function. Tests are used to monitor the products condition during normal operation, to support periodic maintenance activities, and to diagnose faults in trouble situations.

Figure 1 also indicates that products frequently have multiple levels of assembly (e.g.; circuit card, subsystem, system) and test activities occur at all levels within each life cycle phase.

The lower portion of Figure 1 identifies five categories of test information associated with every test application.

— *Design Data*. This includes characteristics of the test subject that must be known before tests can be defined. It can include both static structural properties and dynamic behavior.

— *Test Strategy*. This includes the targeted set of test subject faults, specifications for a set of individual tests, and diagnostic knowledge. The targeted set of test faults is the set of defects that the test application is expected to detect if present. Each individual test is characterized by a measured test subject characteristic, its acceptable range of values, and the method used for making the measurement. Diagnostic knowledge correlates test outcomes with the test subject faults that they reveal. The test strategy within a specific test context contributes to the test requirements that must be defined and met.

— *Test Control*. This consists of manual or automatic test procedures for the test subject.

— *Test Resources*. This includes descriptions of test equipment configurations, capabilities, and control characteristics.

— *Test and Maintenance Information*. This consists of historical data collected during performance of tests. It can include measured values for each individual test, recommended corrective actions, and maintenance actions that were performed. This collected information can be used within and between various phases of the product life cycle to assure proper maturation of the testability and diagnostics of a product, as well as the product itself.

The lower portion of Figure 1 also shows several paths where improved flow of test information can reduce the overall cost of test. The solid arrows represent information flow dependencies during the development process and the subsequent feedback of actual test results within a single test application. However, test information is rarely in a form that other, related test applications can readily access and use (e.g.; by integrated diagnostics). The dotted arrows indicate that frequently the transfer of test information from one level of system integration to another is nearly nonexistent.

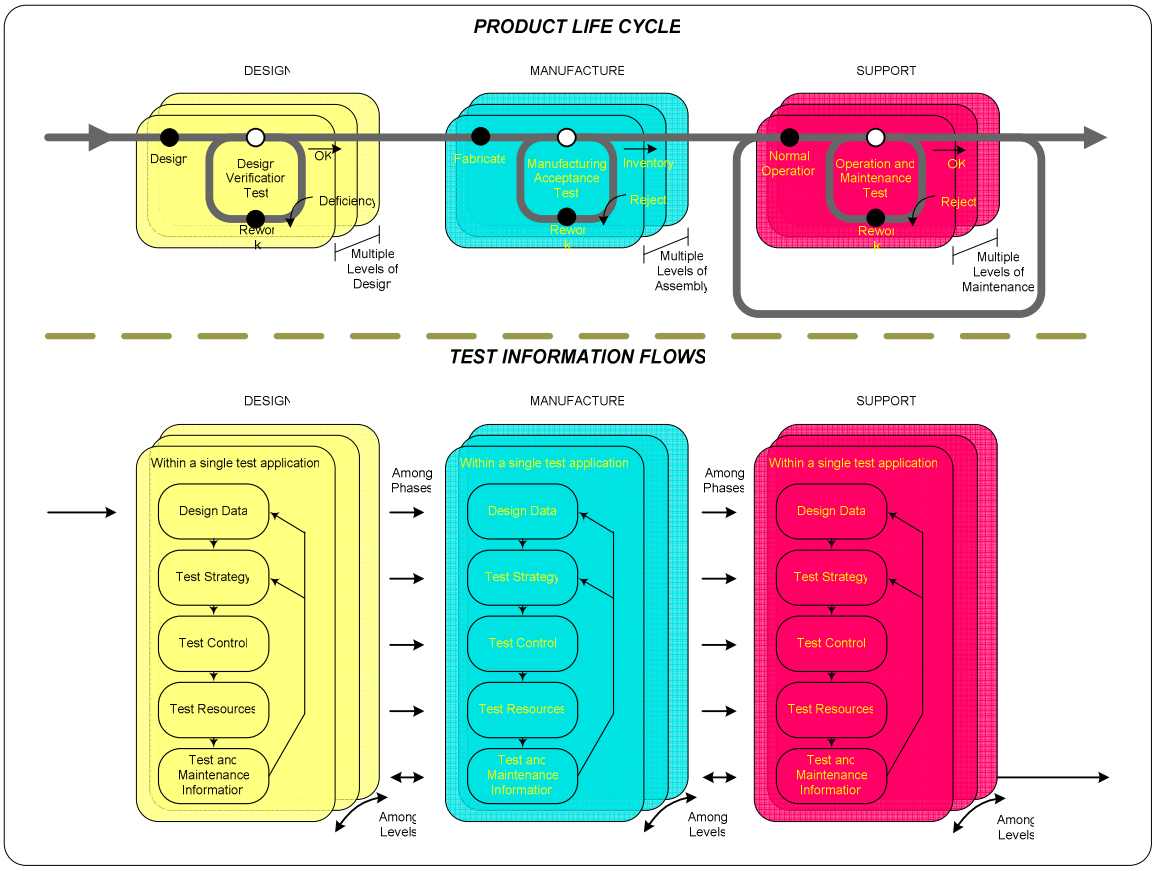

Figure 1—Product Life Cycle and Test Information Flow

The use of different test systems and propriety data formats has inhibited the exchange of useful test information. ATML, by defining standards for representing test information, promotes its exchange among diverse test environments. In addition, ATML promotes the feedback of test and maintenance information, enabling continuous improvement of both the product and its test processes. This is accomplished by strengthening the interfaces between activities and phases of the product life cycle, thus promoting free flow of required information. Results of experience supporting a product can be used to recommend changes in product design, which would be fed forward to the design phase for the next generation of the product (indicated by the start of a new product life cycle).

It may not always be practical to completely reuse test information in subsequent life-cycle phases. The amount of reuse that can be achieved depends on the context within which the testing occurs, the extent to which common test information applies, and the availability of test resources with the required capabilities. Recognizing these limitations, ATML organizes test information in a way that allows the useful elements to be found and reapplied. The effort needed to capture a sufficiently rich set of test information is an investment that is recouped when the pertinent information is found and reused in other test applications.

4.3.1 Design

During the design phase design verification tests are often performed to verify that products built according to the design will satisfy the established functional and structural specifications. Design verification tests are especially important prior to entering a large production phase. Tests are distinguished for other design verification methods such as inspection or analysis. In one approach to design verification, tests are performed using a prototype or preproduction model built according to the design. These tests generally use instrumentation to acquire actual data and compare it to acceptance criteria. Nominal performance tests

verify product operation under typical operating conditions. Stress tests verify that the product continues to perform acceptably when one or more external conditions approach their limits.

The use of virtual test methodologies is also supported by ATML through the specification of virtual resources whose operation can be simulated, thereby providing for test simulation in design environments. This capability can be used to help evaluate test strategies for subsequent phases and capture the results for later use during actual implementation. IEEE Std. 1641-2005 (Standard for Signal and Test Description) provides for this simulation specification, and is intended to be used as the common signal reference for use throughout the life cycle of the UUT or test system.

The best opportunity to reduce the costs of tests over the entire product life cycle occurs during the design phase. Cost savings can be achieved by adequate planning for test applications in the subsequent life-cycle phases. This planning includes incremental testability analysis, evaluation of alternative maintenance strategies, consideration of available test methods, and assessment of available or planned test equipment. Testability in later life cycle phases is often constrained by decisions made during design. As shown in Figure 1, the information generated during design should be captured so that it can be fed forward to the other product life-cycle phases. These later test activities need access to comprehensive product information, including functional and structural specifications, failure modes and effects, and reliability data. Planned manufacturing and maintenance test strategies should also be captured.

Each phase includes a historical data-gathering activity, indicated by the “test and maintenance information” block that appears in every phase shown in Figure 1. The information collected during the subsequent phases can be fed back to the designers and planners, who can use this information to determine whether design changes are needed to alleviate any test problems. This is indicated by the two-way flow of test and maintenance information between phases and the feedback from test and maintenance information to design information with the phases shown in Figure 1.

4.3.2 Manufacturing

One primary purpose of manufacturing tests is to detect product defects resulting from anomalies in the manufacturing process. This verification process includes in-process tests as well as inspections. At each level of assembly, tests may be used to screen components according to quality-assurance criteria. Final acceptance tests may be performed prior to product delivery. Another purpose of some manufacturing tests is to characterize components. For example, items such as resisters can be placed into separate bins depending on required accuracy ranges.

The manufacturing test process for a product is determined in accordance with the manufacturer’s capabilities, equipment, budget, and policies. At successfully higher levels of assembly (e.g.; circuit card, subassembly, system), manufacturing tests are expected to detect faults introduced during the assembly process. When manufacturing tests are based on a functional test strategy, access to design verification test information can significantly reduce test development costs. It is likely that manufacturing test developers will be able to reuse significant parts of the design verification test strategy. Test strategies for subassemblies can also be integrated with the test strategies at higher levels to ensure that all targeted faults are covered and consistent limits used to ensure that units will operate properly when installed in higher-level assemblies. Assuming that design deficiencies have been corrected, recurring failures in manufacturing tests are indicative of anomalies in the fabrication process.

4.3.3 Support

During the support phase, tests are typically used when a system failure occurs or scheduled maintenance is performed. If a failure occurs, a maintenance test can be used to isolate the fault, leading to repairs. Tests performed as part of scheduled maintenance include tests performed periodically for operational readiness, system calibration, prognostics (e.g.; evaluating the remaining life of a component), and for preventive maintenance purposes.

Field test and repair frequently emphasize expediency. Maintenance tests are used to isolate the fault to a field replaceable unit (FRU), which is replaced with a spare. If he maintenance test passes after the unit is replaced, the system is returned to service. The unit removed from the system is either discarded or returned to the factory or a designated maintenance/repair center.

Sometimes fault mitigation strategies are used, such as when a replacement unit is of lower capability than the unit it replaces. For example, a flat tire on an automobile is usually fixed in the field by replacing the flat with a spare. The spare may be undersized or of lower quality. In this case, the repair is not complete until the original tire has been cycled through a repair center (garage).

When prescribed maintenance procedures do not fully accomplish the repair, additional ad hoc methods are sometimes used to locate the anomaly that is causing the failure. If all this fails to rectify the problem, the system may be returned to the factory or maintenance/repair center, where additional resources are available to accomplish the test and repair.

A maintenance test can be used for both diagnostic and verification purposes. The only distinction is that additional tests may be performed for diagnostic purposes, in order to isolate the fault to a smaller ambiguity group. These additional tests may also require manual operations such as removing access panels and probing that may not be required for verification purposes.

Test and maintenance information from the field is needed by maintenance/repair centers. Information about system failures, associated symptoms, the conditions of failure, and the field test results can be used to help isolate faults. The information can be integrated with reports from other units (and other sites). This consolidated information provides useful input for recognizing recurring problems, and possibly lead to changes in the product design, manufacture, operation, or maintenance process.

Ideally, diagnostics performed in the field will correctly locate the failing subassembly. However, not all diagnostics achieve this objective. As a result, some units returned to a repair center as faulty are actually good and some units that test as OK are actually faulty. Further, it is possible that inadequate system testability limits the ability of the diagnostics to isolate faults adequately. Further, poor maintenance processes and technician training can introduce high levels of human error, leading to incorrect diagnostics. Access to the field test and maintenance information can help maintainers to track and better manage these situations. Historical data for faults modes and reliability data are very useful developing efficient maintenance test strategies.

When there are multiple levels of maintenance, sharing test and maintenance information among the levels is very important. An integrated diagnostic strategy that consistently address targeted faults and test limits during the test development process helps reduce later occurrences of “can not duplicate” and “retest OK” problems that plague many maintenance organizations. Access to both design verification and manufacturing functional test information can help to significantly reduce maintenance test development costs. Access to test and maintenance results from the field enable maintenance/repair centers to more efficiently isolate faults and accomplish repairs.

4.4 ATML Framework

A framework is a reusable object-oriented design expressed as a set of abstract classes and the way their instances collaborate. In the context of ATML, the abstract classes are component standards representative of all or part of an ATS. The framework provides a context for the components to be used.

The ATML framework has been developed to:

— Summarize and organize the essential elements of an ATS.

— Provide a common frame of reference.

— Eliminate the need to use a variety of custom file formats.

— Provide compliance with the W3C standards.

— Be standards based.

— Be extensible.

— Be pluggable/modular (components based upon the ATML component standards can easily be substituted and data can be shared between the components).

The ATML component standards are the core elements of the ATML framework. Figure 2 portrays the integrated family of standards that make up the ATML framework in the yellow shading.

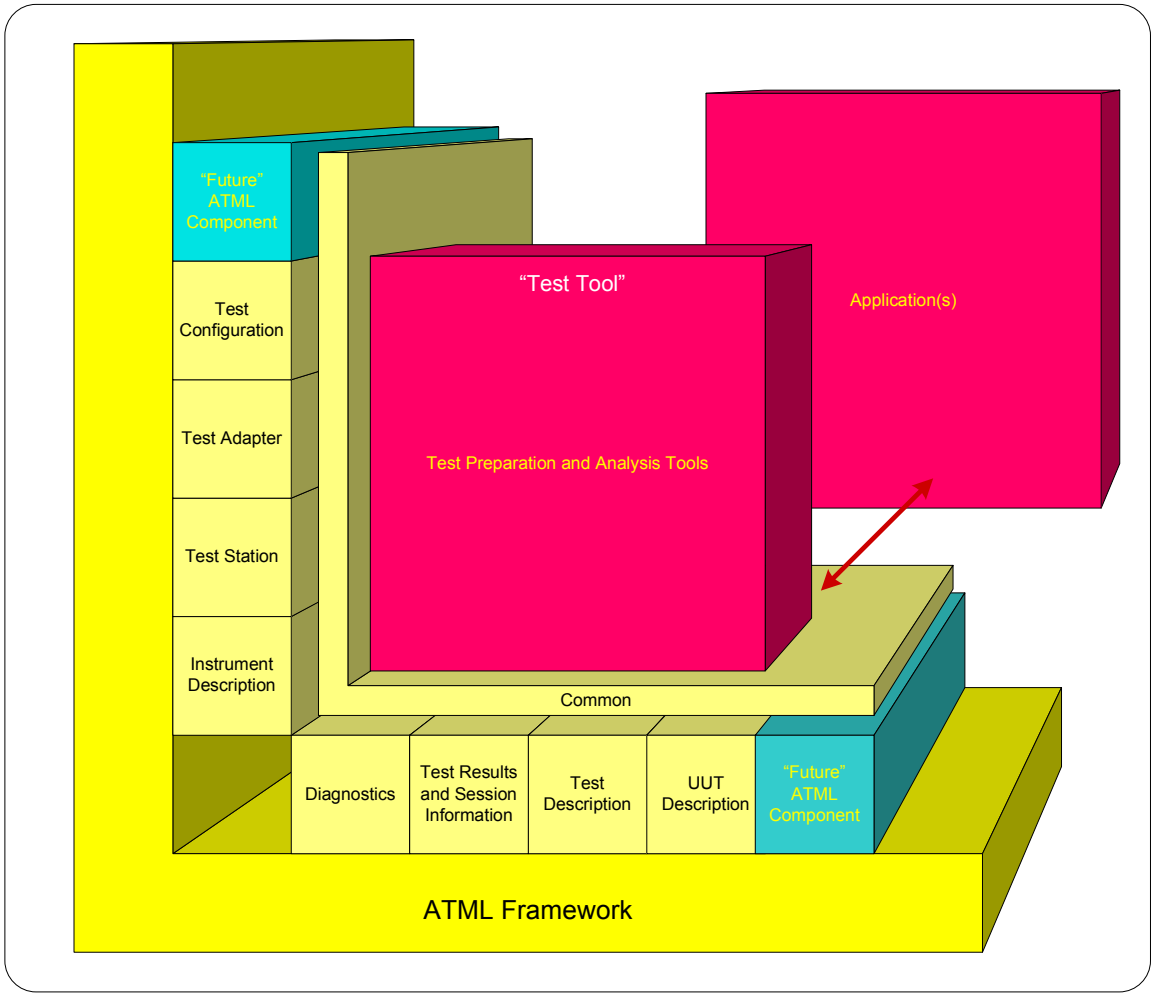

Figure 2—ATML Framework, Components, and Compliant Applications

The test subject standards, represented by the horizontal axis of Figure 2, support the capture and reuse of test subject information. Opportunities for reuse of test subject information include situations where products are tested in various phases of the product life cycle and situations where common components are used in multiple products. Test subject information captures specifications for test subject design and test requirements, which avoids rediscovery efforts in the initial development, maintenance, and re-hosts to test applications. The test subject information also includes diagnostic knowledge that can be accessed during the test process, such as recommended corrective actions based upon the test results.

Test resource standards, represented by the vertical axis of Figure 2, apply to test resource control and information. Test resource standards support specifications for test application resource requirements and

test equipment capabilities. These standards support adaptation of test applications to changes in test equipment configurations, which can range from the replacement of a single instrument to re-host in a separate test environment.

As previously stated, the ATML framework is defined in the form of ATML Components. The ATML Components define the domain-orientated information. ATML Components are segmented into:

— ATML Component Standards (in the form of formal IEEE Standards).

— XML Schemas.

— Reference to ATML Instance Documents. (however, specific instance documents are not part of the ATML framework)

4.4.1 ATML Component Standards

The ATML Components (as denoted in Table 1) have an associated IEEE published Standard. Each of the ATML Component Standards contains the definition, description, and use of each element of the ATML Component.

Each ATML Components published IEEE Standard can be acquired from the IEEE as a published work. As such, United States and European Union Copyright Laws restrict the use of their content without appropriate license agreements with the IEEE.

4.4.2 XML Schemas

A majority of the ATML Component Standards have an associated XML Schema (as denoted in Table 2). XML Schemas are described in Clause 4.5.1.

XML Schemas are to be located on the World Wide Web at the locations defined in Clause 4.7.

The IEEE-SA, IEEE Computer Society, SCC20, and the Creative Commons Organization are, as of the date of this publication, addressing the acquisition, use, and licensing of XML Schemas associated with IEEE Standards. Upon resolution and guidance from the IEEE-SA, this clause will be updated.

Figure 3 illustrates example ATML Component Standard content alongside the associated XML, where the ATML Component Standard and XML Schemas can be obtained, as well as what is copyrighted material.

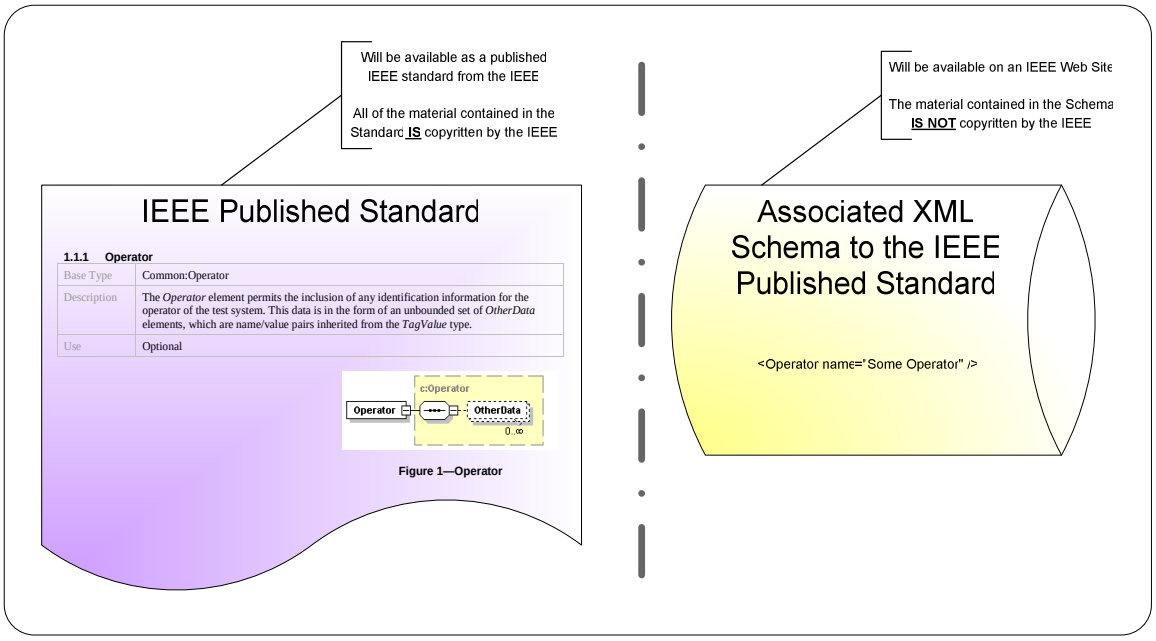

Figure 3—IEEE Standards and XML Schemas

4.4.3 ATML Instance Documents

ATML instance documents are a collection of specific information defined and organized by the referenced XML schema. (e.g.; a "widget" instruments instance document shall contains the definition of the "widget", per the instrument description XML schema specification)

As such, the individual ATML instance documents are not part of ATML framework standardization.

4.5 Specification Techniques

The use of XML to define the ATML framework is discussed in the following sub-clauses.

4.5.1 XML Schemas and their use in ATML

The XML Schema is the formal language used to specify information requirements. The language focuses on the definition of entities, which are the objects of interest. The entities are defined in terms of their elements and attributes, which are the traits or characteristics considered to be important for using and understanding them. These elements or attributes have a representation, which might be a simple data type (such as integer) or another entity type. The XML Schema also specifies constraints, rules, and relationships between entities.

ATML uses XML Schemas to precisely specify the data that can reside in an ATML test environment. XML Schemas are specified for those categories of test information where different sets of data can be instantiated and exchanged between ATML implementations. Test information that conforms to the ATML Schemas can be accessed and manipulated by software tools in an ATML test environment. A set of best practices and guidelines for the development of ATML XML schemas is provided in Annex A of this Standard.

The advantage of XML Schemas is that it permits unambiguous specification of knowledge that can be shared by different standards or information processing systems.

4.5.2 Relationships between ATML and the IEEE AI-ESTATE, SIMICA, & STD Standards

Certain ATML Component Standards (and their associated XML Schemas) utilize or reference capabilities defined in existing IEEE Standards and projects under development. Specifically, ATML Component Standards utilize or reference the following IEEE standards:

— IEEE Std. 1641 (Standard for Signal and Test Definition): This Standard provides the means to define and describe signals used in testing.

— IEEE Std. 1232 (Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments): This Standard provides the means to exchange and process diagnostic information, as well as control the diagnostic processes (processes such as: testability analysis, diagnostic reasoning, diagnosability assessment, maintenance support, or diagnostic maturation).

— IEEE P1636 (Standard for Software Interface to Maintenance Information and Collection Analysis): This Standard provides the means to store and retrieve machine processable representations of historical diagnostic and maintenance information.

4.5.2.1 IEEE Std. 1641 (STD)

ATML Diagnostics, Test Description, Instrument Description, Interface Adapter, and Test Station all utilize the concept of a signal description within the Standards specification and within the XML schema. Where this occurs, the IEEE Std. 1641 construct(s) will be referenced. ATML Component Standards do not redefine, repeat, or compete with the constructs defined in IEEE Std. 1641. Should any ATML signal description requirement not be satisfied by the STD Standard, the requirement(s) shall be brought to the IEEE SCC20 TAD subcommittee in the form of a change proposal to the STD Standard.

4.5.2.2 IEEE Std. 1232 (AI-ESTATE)

ATML Diagnostics requires a means to exchange and process diagnostic information, as well as control the diagnostic process. All of these ATML Diagnostic requirements will be supported by the utilization of the AI-ESTATE Standard. Should any ATML Diagnostic requirement not be satisfied by the AI-ESTATE Standard, the requirement(s) shall be brought to the IEEE SCC20 DMC subcommittee in the form of a change proposal to the AI-ESTATE Standard (ATML Diagnostics shall not become a “competing standard” to AI-ESTATE).

4.5.2.3 IEEE P1636 (SIMICA)

ATML Diagnostics requires a means to store and retrieve historical diagnostic and maintenance information. All of these ATML Diagnostic requirements will be supported by the utilization of the SIMICA project Standard. Should any ATML Diagnostic requirement not be satisfied by the SIMICA project Standard, the requirement(s) shall be brought to the IEEE SCC20 DMC subcommittee in the form of a change proposal to the SIMICA project Standard (ATML Diagnostics shall not become a “competing standard” to SIMICA).

4.6 ATML Component Standards

The family of ATML standards (with its associated XML schemas) defines a logically related set of ATML information. These ATML component standards elaborate on information that only appears as "place holders" in this standard. The entire ATML family of standards consists of:

Table 1—ATML Components

Component Name	Brief Description	Standard
Standard Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML	This Standard	IEEE Std. 1671 (ATML)
Common	Contains the shared type definitions utilized within two or more ATML components	None
Diagnostics	Supports the execution and analysis of diagnosis and diagnostic procedures	IEEE Std 1232 (AI-ESTATE) and IEEE P1636 (SIMICA)
Instrument Description	Provides for the description of an test instrument	IEEE P1671.2 (ATML: Instrument Description)
Test Adapter	Provides for the description of an interface test adaptor (ITA)	IEEE P1671.5 (ATML: Test Adapter)
Test Configuration	Provides for the description of the testing configuration	IEEE P1671.4 (ATML: Test Configuration)
Test Description	Provides for the description of the test subjects test requirements	IEEE P1671.1 (ATML: Test Description)
Test Results and Session Information	Contains the results of a single run of an test, or tests, performed on a test subject	IEEE P1636.1 (SIMICA: Test Results and Session Information)
Test Station	Provides for the description of a test station	IEEE P1671.6 (ATML: Test Station)
UUT Description	Provides for the description of a test subject	IEEE P1671.3 (ATML: UUT Description)

4.6.1 Common

Common provides the definition of common types and attributes used by more than one of the XML schemas. This XML schema is simply is a "toolbox" for the other XML schemas.

Common is not a stand-alone IEEE standard.

4.6.2 Diagnostics

Diagnostics facilitates diagnostic information sharing to support the execution and analysis of diagnosis and diagnostic procedures.

While ATML Diagnostics is strongly focused around Artificial Intelligence type applications, the ATML definition of diagnostic data is broad enough to not preclude some other maintenance related standards from being developed at some future date.

The process of diagnosis will include information from potentially all of the ATML components. The specific diagnostic information for the Diagnostics component is found in IEEE Std. 1232 Standard for

Artificial Intelligence Exchange and Service Tie to All Environments (AI-ESTATE), and IEEE P1636 Software Interface for Maintenance Information Collection Analysis (SIMICA).

4.6.3 Instrument Description

Instrument Description facilitates the definition of the static description of an Instrument. The Instrument Description will facilitate the description of synthetic / virtual / composite instrumentation.

4.6.4 Test Adapter

Test Adapter facilitates the unique description of the interface between the UUT and the Test Station, the physical and electrical characteristics, the capabilities / performance, the identification and classification, the intended test platform, and the TPS(s) supported, etc. This includes the cables, connectors, wires, contacts, etc.

4.6.5 Test Configuration

Test Configuration facilitates the identification all of the hardware, software, and documentation necessary to test a UUT on a particular ATS.

4.6.6 Test Description

Test Description facilitates the definition of the test performance, test conditions, diagnostic requirements, and support equipment to locate, align, and verify proper operation of a UUT.

Any signal descriptions within a Test Description will utilize the IEEE Std. 1641-2004 Signal and Test Definition (STD) capabilities.

Test Description(s) will be utilized in the development of a TPS.

The history of re-hosting a TPS between ATE will be replaced by re-hosting test descriptions. Test descriptions can be implemented on a target ATE by developing a TPS from the test descriptions, in the programming language of choice.

4.6.7 Test Results and Session Information

Test Results and Session Information provides the definition for the data collected that resulted from executing test(s) of a UUT via a TPS in an automated test environment. This includes the measured values, pass/fail results, and accompanying data including test operator, station information, environmental conditions, etc.

Test Results and Session Information is a component standard of IEEE P1636 Software Interface for Maintenance Information Collection Analysis (SIMICA).

4.6.8 Test Station

Test Station facilitates the specification of a particular automatic test station. This includes the physical and electrical characteristics, the paths between test system ports and the Instruments, tolerances and accuracy of the test station, test station identification information such as part number, serial number, nomenclature, location; status information such as calibration data, dates, and self test status; operational history, such as system up-time, external interfaces, safety information such as interlocks, temperature sensing; power requirements controller definitions, etc.

4.6.9 UUT Description

UUT Description facilitates the unique description of a particular UUT. This includes information such as the name, part number, model number, type, power requirements, interfaces, physical properties, and operational requirements.

4.7 XML Schema Names and Locations

The ATML family of standards, their associated XML schemas names, and where each of the XML schemas can be located, are:

Table 2—XML Schema Names and Locations

Component Standard	XML Schema	XML Schema Namespace
ATML (This Standard)	None	None
Common	Common.xsd	http://www.ieee.org/atml/<*release year*>/Common
Diagnostics	CommonElementModel.xsd	http://www.ieee.org/1232/<release year>/CommonElementModel
	DynamicContextModel.xsd	http://www.ieee.org/1232/<release year>/DynamicContextModel
	DynamicInferenceModel.xsd	http://www.ieee.org/1232/<release year>/DiagnosticInferenceModel
	EnhancedDiagnosticInferenceModel.xsd	http://www.ieee.org/1232/<release year>/EnhancedDiagnosticInferenceModel
	FaultTreeModel.xsd	http://www.ieee.org/1232/<release year>/FaultTreeModel
Instrument Description	InstrumentDescription.xsd	http://www.ieee.org/atml/<release year>/InstrumentDescription
Signal and Test Definition	stdbsc.xsd stdtsf.xsd	STDBSC STDTSF
Test Adapter	TestAdapter.xsd	http://www.ieee.org/atml/<*release year*>/TestAdapter
Test Configuration	TestConfiguration.xsd	http://www.ieee.org/atml/<*release year*>/TestConfiguration
Test Description	TestDescription.xsd	http://www.ieee.org/atml/<*release year*>/TestDescription
Test Results and Session Information	TestResults.xsd	http://www.ieee.org/1636/<*release year*>/TestResults
Test Station	TestStation.xsd	http://www.ieee.org/atml/<*release year*>/TestStation
UUT Description	UutDescription.xsd	http://www.ieee.org/atml/<*release year*>/UutDescription

In addition to the above schema, additional schemas may be defined if information beyond a serial number is needed when describing an instance. This standard will be updated as new schemas are created.

5. Conformance

Conformance to ATML can be achieved by conforming to one or more ATML component standards, assuming the conformance requirements of the ATML component standard(s) have been satisfied.

An instance document is considered to be conformant if it has been validated by the XML schema and adheres to all requirements specified in the relevant ATML Component Standard.

6. Extensibility

The provision of an extension mechanism is necessary to ensure the viability of the specification and allow producers and consumers of ATML instance documents to interoperate in those case where there is a requirement to exchange relevant data that is not included in the standard XML schema. The use of the extensions should be done in a way ensures that a conformant consumer can utilize the extended file without error, discard or otherwise sidestep the extended data, and use the non-extended portions of the data as it is intended - without error or loss of functionality.

Extensions should be additional information added to the content model of the element being extended. They should not repackage existing information in the XML document.

An extended instance document should be accompanied by the extension XML schema and documentation sufficient to explain the need for the extension as well as the underlying semantics and relationship(s) to the base schema.

ATML schemas support two forms of extension:

— Wildcard-based extensions allow for the extension of ATML schemas with additional elements.

— Type derivation allows for extending the set of data types by deriving a new type from an existing type.

XML schemas control the location and type of extension allowed. Clause A.9 describes how to specify the extension points for an XML schema.

Annex A

(informative)

XML Schema Style Guidelines

A.1 Naming Conventions

A.1.1 Capitalization Conventions

A.1.1.1 Pascal Case

The first letter in the identifier and the first letter of each subsequent concatenated word are capitalized.

A.1.1.2 Camel Case

The first letter of an identifier is lowercase and the first letter of each subsequent concatenated word is capitalized.

A.1.1.3 Uppercase

All letters in the identifier are capitalized.

A.1.1.4 Lowercase

All letters in the identifier are lowercase.

A.1.2 Naming Guidelines

— Spell words using correct spelling. Avoid abbreviations and acronyms.

  — As a general rule, acronyms SHOULD NOT be used in XML element and attribute names. When it is necessary to use an acronym, acronyms with three or more characters use **Pascal case**. Acronyms with two characters use **Uppercase**.

  — Abbreviations MUST NOT be used in XML element and attribute names.

  — For XML schema data types, abbreviations MUST be avoided while acronyms SHOULD NOT be used.

— XML element and XML schema data types use **Pascal case**.

— Except for XML schema abstract data types, XML schema data type names SHALL NOT have the word ‘Type’ appended.

— XML attributes use **Camel case**. There is one exception to this rule. If an element has an “ID” attribute, that attribute should use the **Uppercase** naming convention and be of type **NonBlankString** if it is required, and be of type **xs:string** if it is optional.

— Namespace names use **Pascal case**.

— Namespace prefixes use **Lowercase**.

— Use mixed case instead of underscores to distinguish name segments.

— Underscores, periods and dashes MUST NOT be used in XML element, schema data type, or attribute names.

— An element that represents a collection shall be named using a plural name.

— An element that represents a single entity shall be named using a singular name.

A.2 XML Declaration

All ATML schema and instance documents shall use an explicit XML declaration as the first line of a file. This declaration shall follow the form: *<?xml version opt._encoding opt._standalone?>*. In general, it is expected that all ATML documents will use UTF-8 encoding and will not use the standalone option. Thus, the XML declaration for ATML documents shall be:

```
<?xml version="1.0" encoding="UTF-8"?>
```

A.3 ATML Namespaces

The namespace URL for approved schemas shall be:

`http://www.ieee.org/ATML/<release_year>/<schema_name>`

where

<release_year> is the year in which the schema was approved.

<schema_name> is the schema name identified in Table 2.

The namespace URL for pre-approved (draft, candidate, recommendation) schemas shall be:

`http://www.ieee.org/ATML/<posting_year>/<version>/<schema_name>`

where

<posting_year> is the year in which the pre-release version of the schema is made available

<version> is an integer that indicates the version of the pre-approved schema. The version starts at 01 and increments each a new pre-approved version is made available for evaluation

<schema_name> is the schema name identified in Table 2.

The namespace shall be modified whenever one of the following conditions occurs:

A change is made to a schema that invalidates existing instance documents (i.e. a major revision update)

The schema's state changes from "pre-approved" to "approved"

A new "pre-approved" version is made available for evaluation

The use of these ATML sections is controlled through their namespace, such that any ATML document refers to the namespace when describing one of these components. e.g.

```
<?xml version="1.0" encoding="UTF-8"?>
<TestStation name="CASS" xmlns="http://www.ieee.org/ATML/2005/TestStation">
    <Configuration>
        <Instrument name="Cable1" xmlns="http://www. ieee.org/2005/ATML/Instrument">
        </Instrument>
        <Instrument name="DMM" xmlns="http://www. ieee.org/ATML/2005/Instrument">
        </Instrument>
    </Configuration>
    <Supports>
```

```
        <Equipment>
            <UUT name="LHGS" xmlns="http://www. ieee.org/ATML/2005/TestRequirements"/>
            <TestAdaptor name="123/45"
                xmlns="http://www. ieee.org/2005/InterfaceAdapter"/>
        </Equipment>
    </Supports>
</TestStation>
```

A.3.1 Target Namespace

Every XML schema should define a target namespace. The namespace should be defined as a URL as described in clause A.3. Each ATML component XML schema has its own namespace. This provides a standard way to avoid name collisions between schemas.

A.3.2 Default Namespace

The default namespace should be the target namespace.

A.3.3 XML Schema Namespace Reference

The namespace prefix for the XML Schema namespace should be **`xs:`**

`xmlns:xs:=”http://www.w3.org/2001/XMLSchema”`

The XML Schema namespace should not be the default namespace.

A.3.4 Qualified and Unqualified

There are two attributes of the **`xs:schema`** element that should be specified for every XML Schema: **`elementFormDefault`** and **`attributeFormDefault`**. These attributes specify whether or not elements and attributes in XML instance documents need to be qualified with the namespace of the XML schema in which they are defined.

The value of **`attributeFormDefault`** specifies whether or not attributes in XML instance documents are qualified with the namespace of the XML schema in which they are defined. Since an attribute is always defined and used in the context of an element, it is not necessary to quality the attribute as well as the element.

The value of **`attributeFormDefault`** should be **`unqualified`**.

The value of **`elementFormDefault`** specifies whether or not elements in XML instance documents are qualified with the namespace of the XML schema in which they are defined. A value of **`qualified`** indicates that if the root element is qualified, then all sub-elements must be qualified as well. A value of **`unqualified`** indicates that only global elements need to be qualified. Using a value of **`unqualified`** allows for inconsistent qualification of elements in instance documents.

Given the following example schema with **`elementFormDefault`** set to **`qualified`**:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:http://mynamespace.com/MySchema
    xmlns:xs=http://www.w3.org/2001/XMLSchema
    targetNamespace="http://mynamespace.com/MySchema"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">

    <xs:element name="GlobalElement">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="ChildElement" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

```
</xs:schema>
```

The following are valid instance documents:

```
<?xml version="1.0" encoding="UTF-8"?>
<GlobalElement xmlns="http://mynamespace.com/MySchema">
    <ChildElement/>
</GlobalElement>
```

and

```
<?xml version="1.0" encoding="UTF-8"?>
<my:GlobalElement xmlns:my="http://mynamespace.com/MySchema">
    <my:ChildElement/>
</my:GlobalElement>
```

The following is NOT

```
<?xml version="1.0" encoding="UTF-8"?>
<my:GlobalElement xmlns:my="http://mynamespace.com/MySchema">
    <ChildElement/>
</my:GlobalElement>
```

If we set **`elementFormDefault`** set to **`unqualified`** in the above schema, then the following are valid instance documents:

```
<?xml version="1.0" encoding="UTF-8"?>
<GlobalElement xmlns="http://mynamespace.com/MySchema">
    <ChildElement/>
</GlobalElement>
```

and

```
<?xml version="1.0" encoding="UTF-8"?>
<my:GlobalElement xmlns:my="http://mynamespace.com/MySchema">
    <ChildElement/>
</my:GlobalElement>
```

The following is NOT

```
<?xml version="1.0" encoding="UTF-8"?>
<my:GlobalElement xmlns:my="http://mynamespace.com/MySchema">
    <my:ChildElement/>
</my:GlobalElement>
```

The value of elementFormDefault should be qualified.

A.4 Versioning

The schema version shall be captured in the schema using the version attribute of the schema element.

The format of the schema version shall be <major>.<minor>, where the major portion shall always begin at ‘0’ and the minor portion shall be a two digit number beginning from “00”.

Previous *released* versions of each schema shall be made available on www.ieee.org.

Changes made to an XML schema fall into two categories:

a) A non-invalidating change – a non-invalidating change is one that does not invalidate existing instance documents. That is, existing instance documents will continue to validate against the new version of the schema. Examples include correcting or adding annotation data, adding an optional element, adding an optional attribute or adding an enumeration item. For this type of change, it is sufficient to increment the <minor> portion of the version.

b) An invalidating change – an invalidating change is one that invalidates existing instance documents, that is, existing instance documents will no longer validate against the new version of the schema. Examples include adding required elements or attributes, changing the structure of an element, or renaming an element or attribute. For this type of change, the <major> portion of the version must be incremented and the minor portion of the version will be reset to zero (00). Also in this case, the namespace of the schema must be changed.

A.4.1 Versioning process for non-invalidating change

a) Change the schema version number within the schema (<minor> portion is incremented by 1).

b) Document the change in the schema change history.

c) Make the new and previous version of the schema available.

A.4.2 Versioning process for an invalidating change

a) Change the namespace

b) Change the schema version number within the schema (<major> portion is incremented by 1, <minor> portion is reset to 00).

c) Document the change in the schema change history.

d) Make the new and previous version of the schema available.

A.4.3 Version process releasing an approved schema

a) Change the namespace to replace `<posting_year>/<version>` with `<release_year>`

b) Make the new and previous version of the schema available.

A.5 Documentation

Use the annotation element for documenting the schema. The **`<xs:annotation><xs:documentation>…</xs:documentation></xs:annotation>`** elements shall contain information targeted at human readers of the XML schema. Use the annotations to capture semantics, definitions and other explanatory information.

A.6 Element versus Type

When in doubt, define a type. Declaring elements with a named type permits reuse.

A.7 Design

A schema shall define at most one global element. A global element is an element declaration that is an immediate child of the <schema> element.

A schema may define one or more global type definitions.

All elements shall be defined using type definitions. This approach maximizes reuse and namespace control.

Avoid the use of global attributes.

A.8 Element versus Attribute

There is no easy answer to the element versus attribute question. As a general convention, elements are the "real" containers of data. Attributes are used to annotate elements with metadata describing the content of the element. Perhaps the biggest advantage of using element content to represent information in the document and using attributes for annotation is extensibility. The decision to use elements versus attributes should never be made to optimize document size.

A.9 Extensibility

An element has an extensible content model if in instance documents that element can contain elements and data beyond that specified by the schema. ATML schemas should explicitly identify where they can be extended. Only elements from a namespace different from the document namespace should be allowed in an extension. The schema shall use the **<xs:any>** element with the **namespace** attribute set to "**##other**" to identify where extension is allowed. To avoid non-deterministic validation, the **<xs:any>** element shall be included in an **<Extension>** or **<Other>** element. This approach, often referred to as Wildcard extensions, is the only approach that allows an extended instance document to validate against the original XML schema definition.

Allowing the extension of a schema using type substitution should be avoided. Schemas should mark elements defined via a simple or complex type with the **block** attribute set to **#all** if type substitution is to be avoided. Elements which use type substitution as their means of definition should be set the **abstract** attribute to **true**.

A.10 Defining Uniqueness and References

When defining a schema for which validation of references is desired, **xs:key** and **xs:keyref** shall be used instead of **xs:ID** and **xs:IDREF**.

When defining a schema for which validation of unique identifiers is desired, **xs:unique** shall be used instead of **xs:ID**.

These requirements arise from the fact that there is no limitation on the values or types that can be used as part of an identity constraint that uses **xs:unique**, **xs:key** and **xs:keyref**, whereas **xs:ID** can only be of a specific range of values (for example, 7 is NOT a valid **xs:ID**). In addition, the scope of **xs:ID** and **xs:IDREF** is the entire document. The scope of **xs:unique**, **xs:key** and **xs:keyref** is the target scope of the **XPATH** expression included in the **xs:keyref** definition.

A.11 Default and Fixed Values

Default and fixed values should NOT be specified for attributes. If a value for an attribute specified with a '**default**' or '**fixed**' value is not supplied in the instance document, XML validation software automatically inserts the default or fixed value(s). Since validation is not required, making values available only when validation is performed should be avoided.

A.12 Collections

A collection is a list of identical items. When specifying a collection, a containing element should be included. The **minOccurs** attribute of the containing element should be set to 1 if the collection is required and 0 if it is optional. The **maxOccurs** attribute of the containing element should always be set to 1, This implies that if the containing element exists, then the collection has at least one item.

The following is an example of the recommended method for defining a collection of identical items:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:http://mynamespace.com/MySchema
```

```
    xmlns:xs=http://www.w3.org/2001/XMLSchema
    targetNamespace="http://mynamespace.com/MySchema"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">

    <xs:element name="MyElement">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Items" minOccurs="0">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="Item" maxOccurs="unbounded"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="OtherElement"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

</xs:schema>
```

The above schema validates the following XML:

```
<MyElement xmlns="http://mynamespace.com/MySchema">
    <Items>
        <Item/>
        <Item/>
        <Item/>
    </Items>
    <OtherElement/>
</MyElement>
```

Further, the **`minOccurs`** and **`maxOccurs`** values of an **`xs:sequence`** element in an XML schema should be set to 1, the default value.

A.13 minOccurs and maxOccurs

The default value for both of these attributes is 1. Do not explicitly set the value of these attributes if the default value is the value to be used.

Annex B

(informative)

Architecture Examples

B.1 Diagnostics

The following example has been created from a real life scenario, to demonstrate the operational benefits of utilizing ATML diagnostic components.

A Weapons Platform at the Operational Level (O-Level) maintenance activity fails a pre mission checkout. The failure is diagnosed using Weapons Platform BIT. An operator removes and replaces several LRUs from the Weapons Platform, and the system then passes BIT. The removed LRUs are then sent to the Intermediate Level (I-Level) maintenance shop for repair.
In the I-Level shop, each of the LRUs is sent to an automatic test station where they undergo a comprehensive full compliment of functional and diagnostic tests. During diagnostic testing, a fault is identified with an SRU in one (1) of the LRUs. The faulty SRU is removed and replaced; the repaired LRU is retested, and passes. The remaining LRUs are tested, each passes testing and is returned to service (with no faults found). The removed SRU is sent to the depot for repair, which in this case is the manufacturer of the SRU.

Once received by the manufacturer, the SRU is sent for diagnosis and repair. After performing a comprehensive full compliment of functional and diagnostic tests on the SRU, the repair technician determines the presence or absence of a fault. The repair technician replaces any faulty components, and if warranted, returns the replaced components to the components manufacturer for fault analyses.

This example illustrates this typical, autonomous operation of the various levels of maintenance.

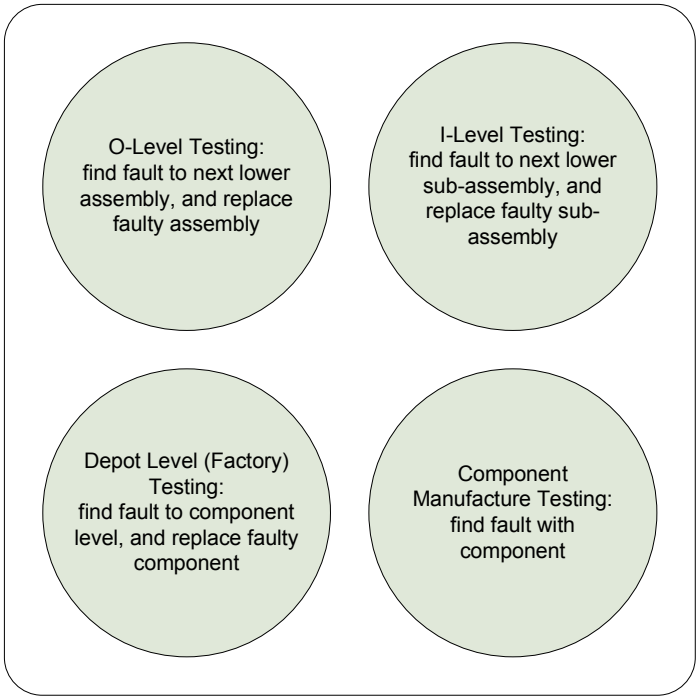

Figure B.1—Autonomous Operation of Maintenance Levels

Using ATML as the means to exchange data between these levels has the potential benefit of reducing the test workload and speeding up the diagnostic process to improve the time spent at each level.

At the O-Level for example, the availability of historical data for the Weapons Platform could reduce the number of pulled and replaced LRUs during the diagnostic process and at the same time reduce the amount of Weapons Platform time required for a particular repair. (e.g.; Bit code 137 calls out LRUs 1, 5 & 7 but the failure, from historical data, 95% of the time LRU 5 is the fault, 4% of the time LRU 7, and LRU 1 less than 1% of the time).

At the I-Level for example, the availability of O-Level data for this occurrence could be used to guide diagnostic testing of LRUs (e.g.; When BIT code 137 calls out LRU 5 the predominant failure is detected 90% of the time in test group 900. The TPS for LRU 5 has a an entry point prior to test group 900, that if testing were to be started there, this would cut 45 minutes off of the detection time of that particular failure mode).

The data collected at all levels of maintenance can be used as verification of the validity or identification of problems associated with maintenance actions as they ripple down the logistics stream as illustrated by figure B.2.

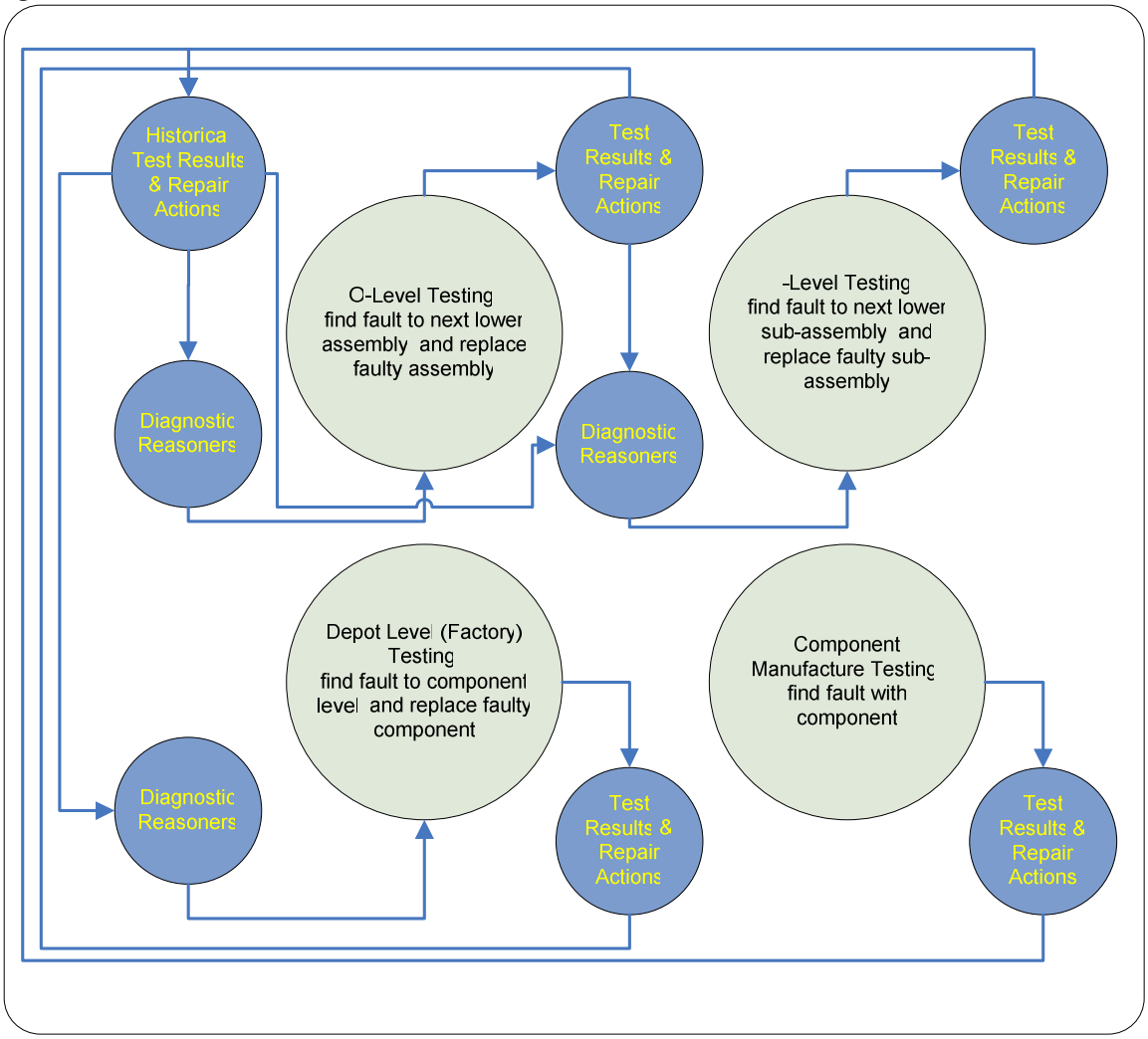

Figure B.2—Example ATML Diagnostics Operation between Maintenance Levels

B.2 Instruments

The following has been created from a real life scenario, to demonstrate the operational benefits of utilizing ATML Instrument Descriptions. The scenario of this example is that of representing instruments capabilities during an ATLAS based TPS compilation process.

Typically, ATLAS based compilation tools utilize some representation of station instrumentation capabilities. This file (or maybe database) usually is of a proprietary format, unique to the particular vendor who developed the compilation toolset. This is then populated with data (either by ATE system integrators, or the tool vendor) which is at best, an interpretation of each of the instruments capabilities, each manipulated to "fit" within the constraints of the proprietary format and/or the function of the compiler.

The philosophy ATML is consciously supporting is to no longer attempt to interpret instrument capabilities, but to use and maintain the instrument data as provided by the manufacturer through utilization of the InstrumentDescription.xsd schema.

It is worthwhile noting, although not applicable to this example, that in addition to the InstrumentDescription.xsd schema that would exist for a particular vendors product (e.g.; Vendor X, Model ABC), that ATML permits an Instrument Description instance schema for the particular serial number and configuration of the actual product "purchased" (e.g.; Vendor X, Model ABC with Option 1A, Serial Number 12,345).

ATML tool developers have already begun enhancing existing tool sets to utilize InstrumentDescription.xsd schemas in lieu of their proprietary databases.

The following example illustrates using the instrument vendor original instrument description rather than interpreting data sheets into yet another specific implementation. ATML implemented properly, makes interpreting instrument datasheets by system integrators and maintainers a "thing of the past".

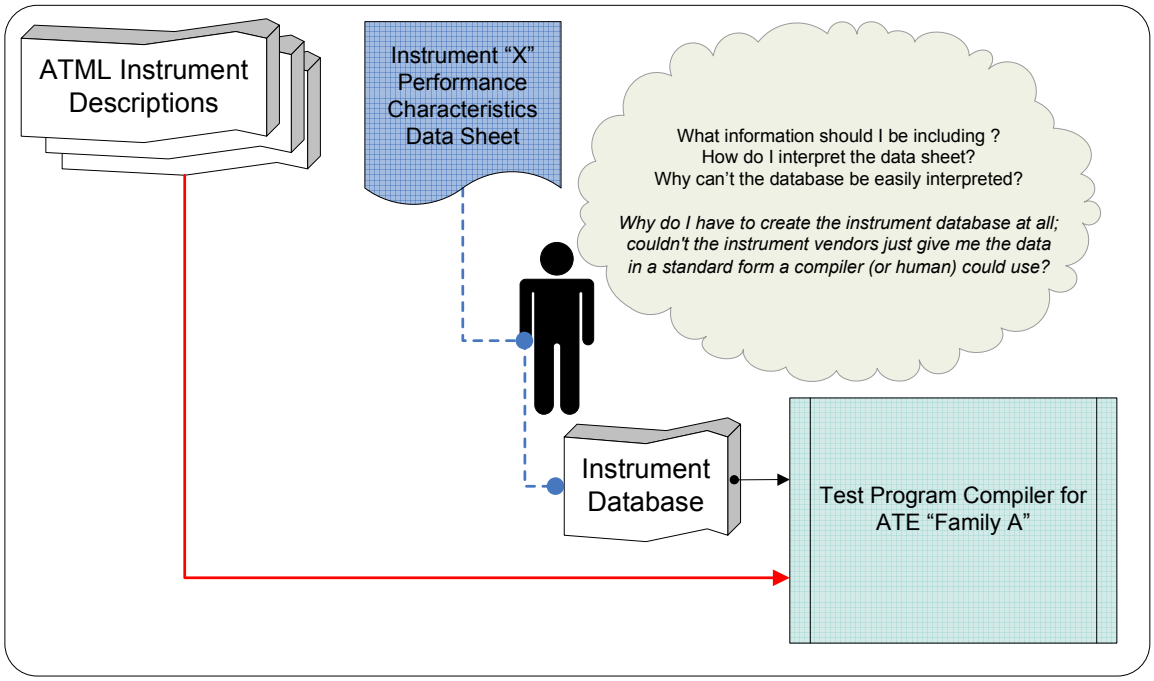

Figure B.3—Example ATML Instrument Usage/Benefits

B.3 Test Descriptions

The following has been created from a real life scenario, to demonstrate the operational benefits of utilizing ATML Test Descriptions. The scenario of this example is that of moving (either by re-hosting or porting) a TPS from one ATE family to another.

Typically, TPSs are developed and integrated with a particular ATE. What this methodology introduces is station dependencies integrated within the TPS. These station dependencies can be categorized as:

a) **TPS instrument programming**: An "historical" programming technique that evolved from the manual testing methodology, and is widely utilized today. The "result" is that a test is written around a particular instrument, its "features", and the instruments operation within that ATE. The "problem" with this methodology is that it is nearly impossible to replace the instrument without some level of modification to the written tests (up to and including a total re-write).

b) **Instrument hardware electrical characteristics**: Electrical characteristics that must be taken into account when interfacing instruments with the UUT. For example impedance, drive capability, resolution, and accuracy – capabilities typically found on instrument data sheets. However there are characteristics that are quite often "overlooked", and not documented, such as timing (e.g.; 3 microseconds after a trigger the waveform is present) or firmware impacts (e.g.; the instrument resets to zero volts between steps of a ramped waveform output). These "overlooked" characteristics quite often have to be reverse engineered, for each instrument and ATE configuration.

c) **Instrument programming interfaces:** The instruments setup time and the time to command the instruments affect the overall TPS execution times as well as individual test timing. Regardless of the interfaces used within the ATE, TPSs were developed with this specific timing integrated within. This timing does not only represent the actual hardware interface, but the controller(s) execution times and software overhead.

It is important to note that at some point in time, the tests that were developed and integrated on a specific ATE, were implementing a test strategy for the particular UUT.

The philosophy ATML is consciously supporting is to no longer attempt to re-use the implemented tests (and address the issues discussed above), but to re-use and maintain the test strategy for the particular UUT. When a new ATE is desired for example, rather than reverse engineering the existing TPS implementation, develop a "new" TPS (in the programming language of choice) targeted to the new ATE (and its capabilities), utilizing the test strategy captured by utilization of the TestDescription.xsd schema.

ATML tool developers have already begun development of tool sets that can extract test requirement schema information from existing ATLAS based TPSs (for use when the test strategy information may no longer be available), as well as support the development of TPS source code from TestDescription.xsd schemas.

The following example illustrates re-using the original test requirement rather than interpreting a specific implementation to create yet another specific implementation. ATML implemented properly, makes re-hosting legacy TPS code a "thing of the past".

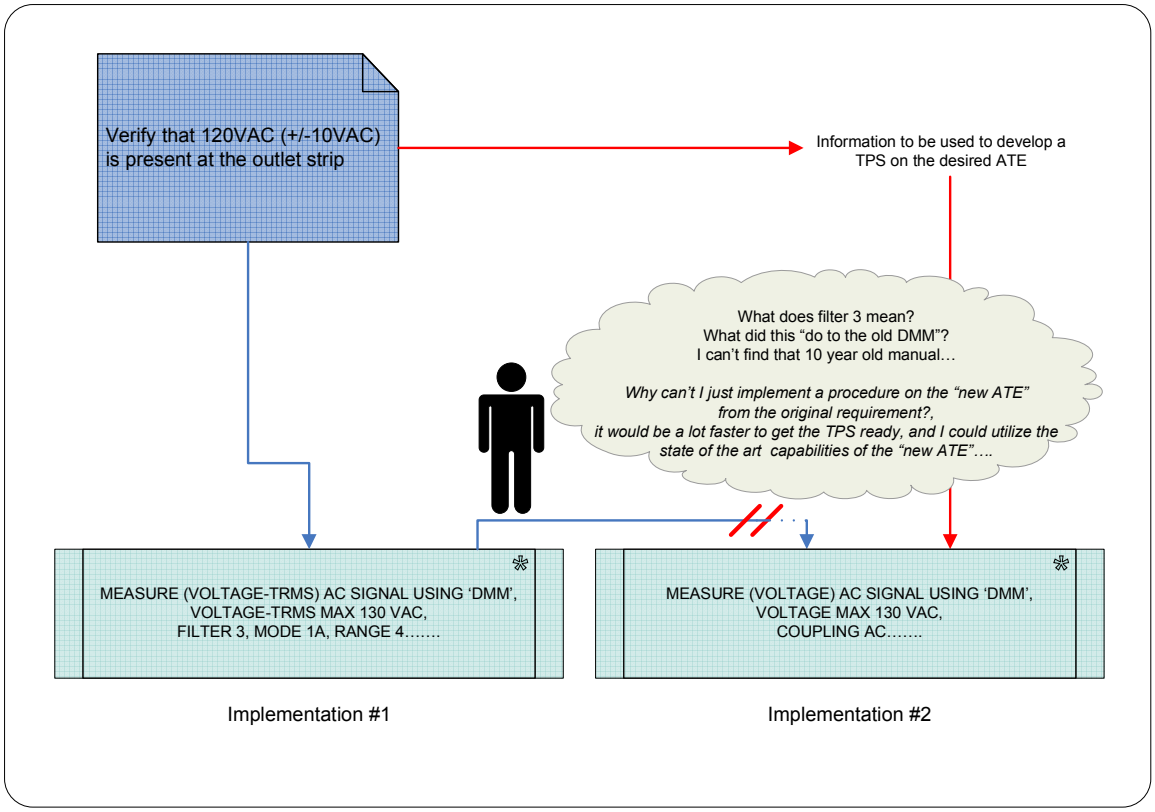

Figure B.4—Example ATML Test Description Usage/Benefits

B.4 Runtime Services

B.4.1 Messages

As a guideline messages and user display information used in an ATML system should utilized well formed HTML for representing their display information. The use of HTML allows the use of standard browser technology to display and interact with the user, in a common format across platforms.

If a requirement exists where ATML documents need to include bits of HTML, and HTML documents to include bits of other markup languages, then use of XHTML should be considered.

B.4.2 Executive System Service

As an example of using the ATML collection of schemas, one classic example would be a service that consumes Test Descriptions and returns Test Results.

The Test Descriptions are an XML document conforming to the ATML schema with the namespace derived from TestDescription.xsd. The Test Results are an XML document conforming to the Test Results and Session Information schema with the namespace derived from TestResults.xsd. Such a service is described in terms of its inputs and outputs; this does not prevent the service providing additionally logging information to internal systems such as an operating system event logger (Windows Event Log or Linux Event Log).

This example assumes an abstract WSDL definition, so that this could be implemented as straight Function Calls, DLL calls, COM Methods, COBRA Services or Web Services.

A WSDL document defines **services** as collections of network endpoints, or **ports**. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: **messages**, which are abstract descriptions of the data being exchanged, and **port types** which are abstract collections of **operations**. The concrete protocol and data format specifications for a particular port type constitutes a reusable **binding**. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service. Hence, a WSDL document uses the following elements in the definition of network services:

- **types**– a container for data type definitions using some type system (such as XSD).
- **message**– an abstract, typed definition of the data being communicated.
- **port Type**–an abstract set of operations supported by one or more endpoints.
  - **operation**– an abstract description of an action supported by the service.
- **binding**– a concrete protocol and data format specification for a particular port type.
- **service**– a collection of related endpoints.
  - **port**– a single endpoint defined as a combination of a binding and a network address.

B.4.2.1 Example WSDL Service Definition

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions
        xmlns=http://schemas.xmlsoap.org/wsdl/
        xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap/
        xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
        xmlns:tdml="http://www.ieee.org/ATML/2005/02/TestDescription"
        xmlns:trml="http://www.ieee.org/ATML/2005/01/TestResults"
        xmlns:y="http://www.ieee.org/ATML/2005/01/RuntimeServices"
        targetNamespace="http://www.ieee.org/ATML/2005/01/RuntimeServices">
        <message name="testDescriptionIn">
                <part name="parameters" element="tdml:TestDescription" />
        </message>
        <message name="testResultOut">
                <part name="parameters" element="trml:TestResults" />
        </message>
        <portType name="ExecutiveRuntimeSystem">
                <operation name="Execute">
                        <input message="y:testDescriptionIn" />
                        <output message="y:testResultOut" />
                </operation>
        </portType>
</definitions>
```

Because this definition does not contain any specific bindings or services it can be used as part of a Web service call or to define a tradition C/C++ function:

```
CString Execute( const CString parameters );
```

With the added constraints that the input and output strings will conform to the relevant ATML schemas.

Annex C

(informative)

Acronyms

AI-ESTATE	Artificial Intelligence Exchange and Service Tie to All Test Environments
API	Application Program Interface
ATE	Automatic Test Equipment
ATLAS	Abbreviated Test Language for All Systems
ATML	Automatic Test Markup Language
ATS	Automatic Test System
BIT	Built In Test
C/ATLAS	Common/Abbreviated Test Language for All Systems
CAD	Computer Aided Design
CC	Creative Commons
CORBA	Common Object Request Broker Architecture
COM	Component Object Module
DLL	Dynamic Link Library
DMC	Diagnostics and Maintenance Control
EDIF	Electronics Design Interchange Format
FRU	Field Replaceable Unit
HTML	HyperText Markup Language
IEEE-SA	Institute of Electrical and Electronics Engineers-Standards Association
ID	Interface Device or IDentifier
IDL	Interface Definition Language
ITA	Interface Test Adapter
I-Level	Intermediate Level
IVI	Interchangeable Virtual Instrument
LRU	Line Replaceable Unit
O-Level	Operational Level
OMG	Object Management Group
RFI	Receiver Fixture Interface
RTO	Run-time Test Object
SCC20	Standards Coordinating Committee 20
SIMICA	Software Interface for Maintenance Information Collection and Analysis
SRA	Shop Replaceable Assembly
SRU	Shop Replaceable Unit
STD	Signal and Test Definition
STEP	STandard for the Exchange of Product data
TAD	Test and ATS Description
TP	Test Program
TPS	Test Program Set
TPSD	Test Program Set Documentation
URI	Uniform Resource Identifier
URN	Uniform Resource Name
URL	Universal Resource Locator
UUT	Unit Under Test
VISA	Virtual Instrument Software Architecture
VPP	VXI *Plug & Play*
VXI	VME (Versa Module Eurocard) Extension for Instrumentation
W3C	World Wide Web Consortium
WSDL	Web Services Definition Language
XHTML	Extensible HyperText Markup Language

XML	Extensible Markup Language
XSD	XML Schema Document

Annex D

(informative)

Bibliography

[B1] An ascetic view of XML best practices. Available from World Wide Web: http://monasticxml.org.

[B2] Extensible Markup Language (XML) 1.0 (Third Edition). World Wide Web Consortium Recommendation 4 February 2004 [cited 2000-05-05]. Available from World Wide Web: http://www.w3.org/TR/2004/REC-xml-2004020.

[B3] ISO 1000:1992, SI units and recommendations for the use of their multiples and of certain other units.[1]

[B4] IEEE 100, *The Authoritative Dictionary of IEEE Standards, Seventh Edition*.[2]

[B5] MedBiquitous XML Schema Design Guidelines. Available from the World Wide Web: http://www.medbiq.org/technology/tech_architecture/xmldesignguidelines.pdf.

[B6] Namespaces in XML. World Wide Web Consortium Recommendation 14 January 1999 [cited 2000-05-05]. Available from World Wide Web: http://www.w3.org/TR/1999/REC-xml-names-19990114.

[B7] Schenck, D. and Wilson, P., *Information modeling the EXPRESS way*, New York: Oxford University Press, 1994.

[B8] TIBCO XML Resource Center: Best Practices. Available from World Wide Web: http://www.tibco.com/software/standards_support/xmlresources/index_best.jsp.

[B9] Uniform Resource Identifiers (URI): Generic Syntax. Internet Engineering Task Force RFC 2396 August 1998 [cited 2000-08-07]. Available from World Wide Web: http://www.ietf.org/rfc/rfc2396.txt.

[B10] URN Syntax. Internet Engineering Task Force RFC 2141 May 1997 [cited 2000-09-28]. Available from World Wide Web: http://www.ietf.org/rfc/rfc2141.txt.

[B11] W3C XML Schema Design Patterns: Avoiding Complexity. Available from the World Wide Web: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnxml/html/xmlschemacomplex.asp.

[B12] XML Developers Guide for government. Available from World Wide Web: http://xml.gov/documents/in_progress/developersguide.pdf.

[B13] XML Information Set World Wide Web Consortium Candidate Recommendation 24 October 2001 [cited 2004-05-05]. Available from World Wide Web: http://www.w3.org/TR/2001/REC-xml-infoset-20011024.

[1] ISO publications are available from the ISO Central Secretariat, Case Postalė 56, 1 rue de Varembė, CH-1211, Genevė 20, Switzerland/.Suissė (http://www.iso.ch/). ISO publications are also available in the United States from the Sales Department, American National Standards Institute, 25 West 43rd Street, 4th floor, New York, NY 10036, USA (http://www.ansi.org/)

[2] IEEE publications are available from the Institute of Electrical and Electronics Engineers Inc., 445 Hoes Lane, Piscataway, NJ 08854, USA (http://standards.ieee.org/)

[B14] XML Linking Language (XLink) Version 1.0. World Wide Web Consortium Candidate Recommendation 27 June 2001 [cited 2004-05-05]. Available from World Wide Web: http://www.w3.org/TR/2001/REC-xlink-2001062.

[B15] XML Path Language (Xpath) Version 1.0. World Wide Web Consortium Recommendation 16 November 1999 [cited 2003-05-12]. Available from World Wide Web: http://www.w3.org/TR/1999/REC-xpath-19991116.

[B16] XML Schema Part 1: Structures. W3C Recommendation, 2 May 2001 [cited 2004-05-05]. Available from World Wide Web: http://www.w3.org/TR/2001/REC-xmlschema-1-20010502.

[B17] XML Schema Part 2: Datatypes. W3C Recommendation, 2 May 2001 [cited 2004-05-05]. Available from World Wide Web: http://www.w3.org/TR/2001/REC-xmlschema-2-20010502.

[B18] XML Schemas: Best Practices. Available from World Wide Web: http://www.xfront.com/BestPracticesHomepage.html.

[B19] XML Schema Tutorial: Available from World Wide Web: http://www.xfront.com.