Reducing Test Program Costs Through ATML-based Requirements Conversion and Code Generation

Lars Lindstrom
National Instruments
Austin, TX, USA
lars.lindstrom@ni.com

Ion Neag
Reston Software
Reston, VA, USA
ion.neag@restonsoftware.com

***Abstract*—Most military and aerospace organizations maintain their test requirements as paper-like forms stored electronically. When test programs need to be created or modified, these documents are often manually referenced, which can be an inefficient and error-prone process. Additionally, because modifications to test program code are sometimes made without updating the corresponding requirements, implementation and documentation tend to diverge as projects evolve, which has an adverse effect on the long-term maintainability of Test Program Sets (TPSs).**

In the past, the lack of an industry-standard data format for test requirements has imposed limitations on the traceability between test results and test specifications. Previous attempts at automating the conversion of analog and mixed-signal test requirements into test programs produced proprietary solutions with limited adoption.

In this paper, we describe an innovative process in which multiple software applications interact through a standard XML format that conforms to IEEE Std 1671.1 Automatic Test Markup Language (ATML) Test Description. The process uses automated test data conversion and code generation to facilitate the initial creation and long-term maintenance of test programs.

Keywords—ATML, IEEE Std 1671.1, TPS, Test Description, Code Generation

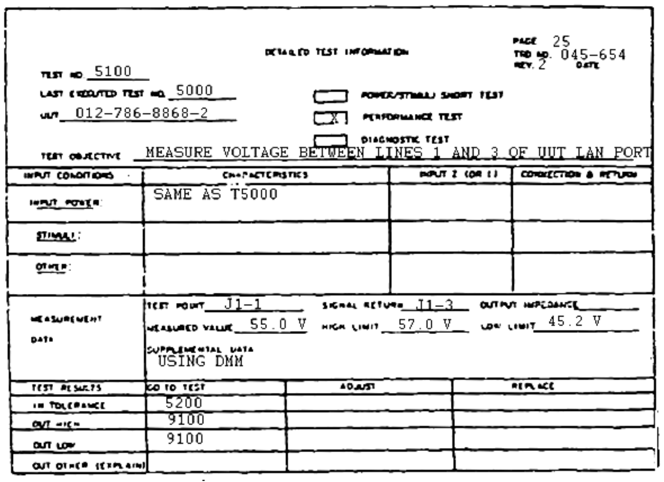
I. INTRODUCTION

In many industries the test requirements for a product are documented in digital documents that are created by product development engineers. When it is time to develop the tests for these units, test engineers convert the tests, limits, and expected behaviors into functional test code. This conversion is often a manual and error-prone process.

By leveraging the IEEE 1671.1 ATML Test Description standard for creating Test Requirement Documents (TRDs), an organization can reduce inefficiencies and ensure interoperability within their organization and their industry. Using this standardization, organizations can also take advantage of commercial-off-the-shelf (COTS) software tools that can import old TRDs, edit existing ones, and automatically convert the TRDs into usable test code.

II. CURRENT PRACTICE

Many new TRDs are created using Microsoft Word, but old TRDs are often available as electronic scans of the original paper documents, like the one shown in Fig. 1.



DETAILED TEST INFORMATION

PAGE 25
TRD NO. 045-654
REV. 2 DATE

TEST NO. 5100
LAST EXECUTED TEST NO. 5000
UUT 012-786-8868-2

[ ] POWER/STIMULI SHORT TEST
[X] PERFORMANCE TEST
[ ] DIAGNOSTIC TEST

TEST OBJECTIVE MEASURE VOLTAGE BETWEEN LINES 1 AND 3 OF UUT LAN PORT

INPUT CONDITIONS	CHARACTERISTICS	INPUT 2 (OR 1)	CONNECTION & RETURN
INPUT POWER:	SAME AS T5000		
STIMULI:			
OTHER:			

MEASUREMENT DATA
TEST POINT J1-1 SIGNAL RETURN J1-3 OUTPUT IMPEDANCE
MEASURED VALUE 55.0 V HIGH LIMIT 57.0 V LOW LIMIT 45.2 V
SUPPLEMENTAL DATA
USING DMM

TEST RESULTS	GO TO TEST	ADJUST	REPLACE
IN TOLERANCE	5200		
OUT HIGH	9100		
OUT LOW	9100		
OUT OTHER (EXPLAIN)			

Fig. 1 – Test Requirements Document Example

TRDs are typically created by product engineers. Test engineers reference the TRDs manually when test programs are created or modified. For example, the test requirements form shown in Fig. 1 could be converted into the following LabWindows™/CVI test code. Note that in this example, we assume that instruments are controlled through a Hardware Abstraction Layer (HAL) that automates switching and wraps instrument commands and driver calls in higher-level functions. Additionally, we assume that test sequencing is implemented separately by using a test executive, such as NI TestStand.

```
ERR_CHK(ConnectInstPort2Pin("DMM_HI","J1-1"));
ERR_CHK(ConnectInstPort2Pin("DMM_LO","J1-3"));
ERR_CHK(dblMeasVal = DMM_Measure());
ERR_CHK(DisconnectInstPort2Pin("DMM_HI","J1-1"));
ERR_CHK(DisconnectInstPort2Pin("DMM_LO","J1-3"));

if(dblMeasVal >= 45.2 && dblMeasVal <= 57.0)
     *nOutcome = PASS;
else
     *nOutcome = FAIL;
```

Fig. 2 further illustrates the current manual requirements entry process.

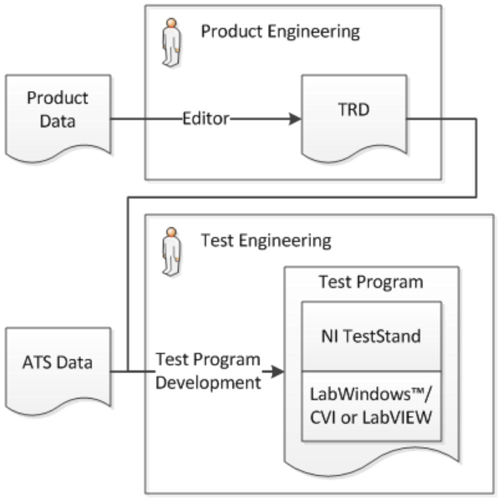


Fig. 2 – Current Practice

The manual conversion of TRDs into test program code has the following disadvantages:

1. Re-typing port names, limits, test numbers, and so on is inefficient and prone to errors.
2. Because code modifications are often made without updating the corresponding TRDs, implementation and documentation tend to diverge, which has an adverse effect on the long-term maintainability of TPSs.
3. The maintenance of TRDs in paper-oriented forms rather than in an electronic data format imposes limitations on the traceability between test results and test specifications.

III. Introducing ATML Test Description

Recognizing the need for electronic data exchange formats in the Automatic Test Systems (ATS) domain, the ATML Working Group and the IEEE developed the ATML family of standards [1]. These standards define electronic data formats based on the XML language. One of the ATML components, IEEE 1671.1 ATML Test Description (TD), defines a data format to represent test requirements [2].

Fig. 3 shows the sections of a typical XML instance document that complies with the ATML Test Description standard. The blocks represent the main sections of an instance document, and indentation represents the hierarchy.

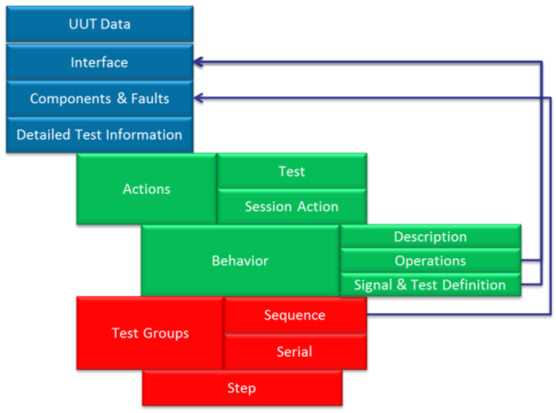


Fig. 3 – Structure of ATML Test Description Documents

The example document includes the following top-level sections:

- **UUT Data**—Contains general UUT attributes, such as part number, revision and manufacturer.
- **Interface**—Describes the electrical interface of the UUT with connectors, pins, and ports.
- **Components and Faults**—Describes the components of the UUT and the failure modes.
- **Detailed Test Information**—Contains data relevant for testing the UUT.

The Detailed Test Information section contains a number of Actions, which can be of type Test or Session Action. Each Action contains a Behavior section, which can contain a free-form Description field, one or more Operation elements, or XML data that conforms to the Signal & Test Definition standard.

The Detailed Test Information section also contains one or more Test Groups, which can be of type Sequence, Serial, and so on. Each Test Group contains a number of Steps.

The arrows in Fig. 3 represent references between sections. For example, Operations of type Connect and Disconnect reference UUT ports defined in the Interface section.

Although the ATML Test Description standard is based on the legacy military TRD standards, it also contains the following significant enhancements:

- Standardized test operations, such as Setup, Measure, Compare, and so on. These types of operations were described as free-form text in the legacy standards.
- Support for describing signals and test behavior using the IEEE Std 1641 Signal & Test Definition standard.
- Test Groups to describe reusable test behavior.
- Multiple types of test sequencing, such as fault tree, serial, and parallel.
- Features that facilitate the accurate exchange of data, such as strongly typed data and support for the specification of standard units.

The following XML snippet is an example implementation of the behavior of the test described by the TRD form in Fig. 1. For brevity, this example does not cover the sequencing of tests or the use of standardized test operations for Connect, Measure, and Disconnect. Readers can download complete XML examples from the IEEE-SA Supplemental Material web site [3].

```xml
<Action xsi:type="Test" ID="816276" name="T5100">
  <Description>Measure Voltage between lines 1 and 3 of UUT
   LAN port</Description>
  <Behavior>
    <Description>Measure DC Voltage between pins J1-1 and J1-3 into
     VoltageValue</Description>
  </Behavior>
  <TestResults>
    <TestResult ID="615736" name="VoltageValue">
      <ValueDescription>
        <DatumDescription xsi:type="doubleDescription" standardUnit="V" />
      </ValueDescription>
      <TestLimits>
        <Limit>
          <c:LimitPair operator="AND">
            <c:Limit comparator="GE">
              <c:Datum xsi:type="c:double" standardUnit="V" value="45.2"/>
            </c:Limit>
            <c:Limit comparator="LE">
              <c:Datum xsi:type="c:double" standardUnit="V" value="57.0"/>
            </c:Limit>
          </c:LimitPair>
        </Limit>
      </TestLimits>
    </TestResult>
  </TestResults>
</Action>
```

The ATML Test Description standard allows the creation of integrated software systems in which software products from different vendors support test requirements input, test document generation, code generation, and so on.

IV. AUTOMATED CODE GENERATION

The solution introduced in this paper and illustrated in Fig. 4 uses ATML Test Description to store test requirements data and replaces the manual conversion of requirements into code by automatic code generation.

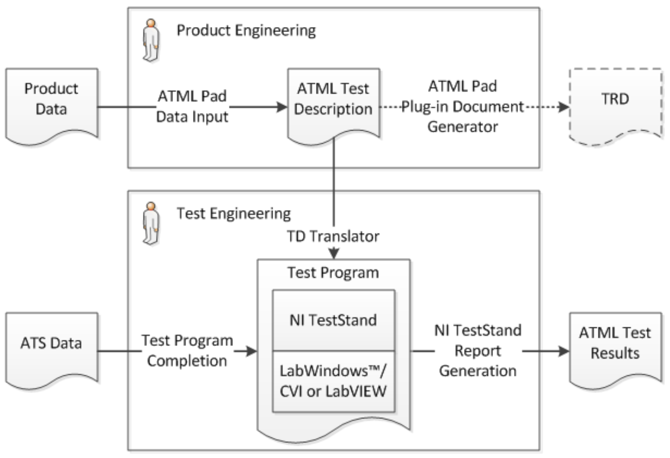


Fig. 4 – Automated Code Generation

A typical test development flow contains the following steps:

1. Product engineers create ATML Test Description documents using ATML Pad, a visual editor for ATML documents [4]. Data are validated to ensure conformance with the applicable IEEE standards and saved directly in the standard ATML Test Description format.

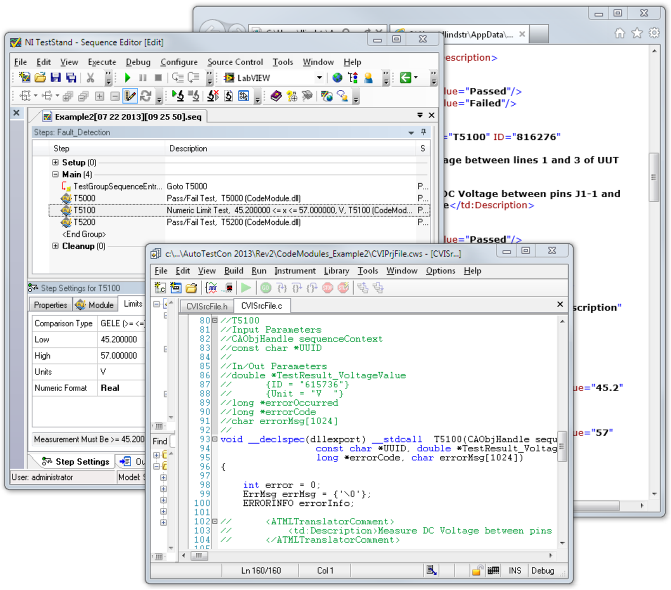
2. If paper-oriented TRDs are required, they can be generated automatically from the ATML Test Description document using a custom plug-in of ATML Pad.

3. The TD Translator (part of the NI TestStand ATML Toolkit) [5] is invoked to perform the automatic code generation, creating a partial TestStand test program that consists of a sequence file and shell source code for LabWindows™/CVI or LabVIEW code modules.

4. Test engineers complete the test program by adding code for instrument control, switching, data processing, operator interface operations, and so on.

When test requirements change, ATML Pad is used to make changes to the ATML document, and then an automatic update function of the TD Translator is invoked to make the corresponding code changes. In most cases, the test program can be changed without altering the code that was added manually since the original translation.

Fig. 5 shows the NI TestStand sequence file and the LabWindows™/CVI shell code that resulted from converting the ATML Test Description file representing the TRD form from Fig 1. To start the conversion, a user simply opens the TEST DESCRIPTION file in NI TestStand and the sequence file and appropriate shell code is automatically created.



NI TestStand - Sequence Editor [Edit]

Example2[07 22 2013][09 25 50].seq

Steps: Fault_Detection

Step	Description
Setup (0)	
Main (4)	
TestGroupSequenceEntr...	Goto T5000
T5000	Pass/Fail Test, T5000 (CodeModule.dll)
T5100	Numeric Limit Test, 45.200000 <= x <= 57.000000, V, T5100 (CodeMod...
T5200	Pass/Fail Test, T5200 (CodeModule.dll)
<End Group>	
Cleanup (0)	

Step Settings for T5100

Comparison Type	GELE (>= <=
Low	45.200000
High	57.000000
Units	V
Numeric Format	Real

c:\...\AutoTestCon 2013\Rev2\CodeModules_Example2\CVIPrjFile.cws - [CVISr...]

CVISrcFile.c

```
//T5100
//Input Parameters
//CAObjHandle sequenceContext
//const char *UUID
//
//In/Out Parameters
//double *TestResult_VoltageValue
//       {ID = "615736"}
//       {Unit = "V  "}
//long *errorOccurred
//long *errorCode
//char errorMsg[1024]
//
void __declspec(dllexport) __stdcall  T5100(CAObjHandle sequ
                    const char *UUID, double *TestResult_Voltag
                    long *errorCode, char errorMsg[1024])
{

    int error = 0;
    ErrMsg errMsg = {'\0'};
    ERRORINFO errorInfo;

//       <ATMLTranslatorComment>
//           <td:Description>Measure DC Voltage between pins
//       </ATMLTranslatorComment>
```

Fig. 5 – Test Sequence Conversion to NI TestStand Sequence

V. Automated Requirements Conversion

The use case referenced in the previous section applies to newly developed test requirements. In many cases, military and commercial organizations already own test requirements, test specifications, or test plans in an electronic document format, such as Microsoft Word, Microsoft Excel, or plain text files. To prevent losing the existing investment, organizations can import existing documents directly into ATML Pad using an extensibility feature called custom plug-ins, as illustrated in Fig. 6.

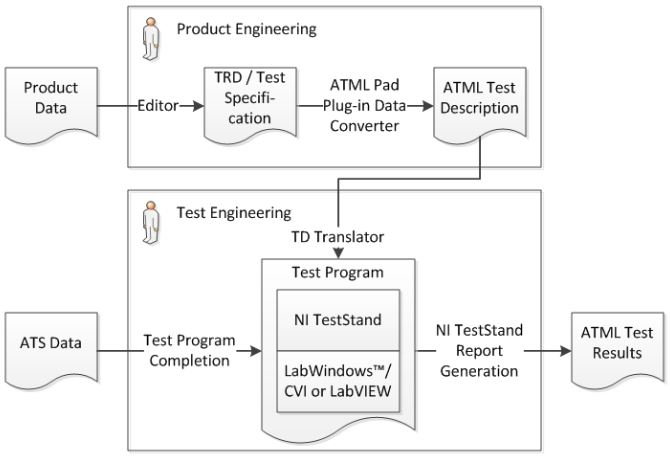


Fig. 6 – Automated Requirements Conversion

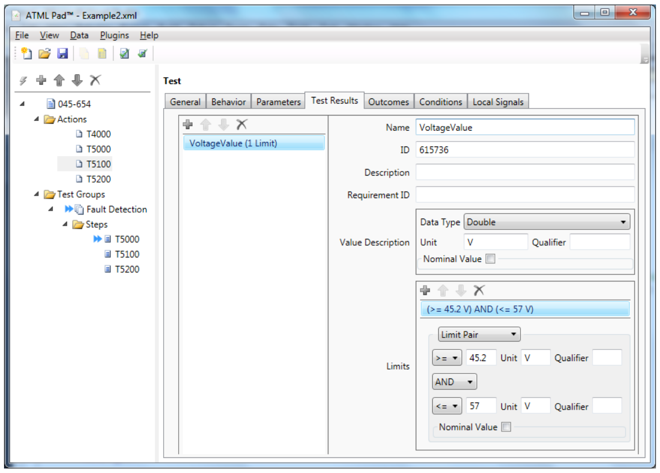
A typical test development flow contains the following steps:

1. Product engineers create TRDs, test specifications, or test plans using a general-purpose document editor. Data are saved in an electronic document format supported by the editor.
2. A specialized plug-in of ATML Pad is invoked to convert the digital document into an ATML Test Description document
3. If necessary, test engineers can edit the generated ATML Test Description document using ATML Pad. For example, free-form descriptions of test behavior can be transformed into standard Operations.
4. The TD Translator performs the automatic code generation.
5. Test engineers complete the test program, as described in the previous section.

VI. ATML Pad

The ATML formats offer powerful modeling capabilities but are often quite complex. Consequently, the use of general-purpose XML editors becomes cumbersome as document size grows. ATML Pad manages the complexity of the ATML format and allows users to focus on describing the test items. ATML Pad includes the following important productivity features:

- Offers an application-specific visual interface that mirrors the structure of the ATML schema. Fig. 7 shows the ATML Pad user interface with the test represented in the TRD form in Fig. 1.



Name	VoltageValue
ID	615736
Description	
Requirement ID	
Value Description	Data Type: Double; Unit: V; Qualifier; Nominal Value
Limits	(>= 45.2 V) AND (<= 57 V); Limit Pair; >= 45.2 Unit V Qualifier; AND; <= 57 Unit V Qualifier; Nominal Value

Fig. 7 – User Interface of ATML Pad

- Supports efficient data input. For example, users can quickly generate complex ATML constructs through a single mouse click.
- Abstracts XML ID references, allowing users to select the referenced item from a list.
- Generates XML IDs automatically and ensures that IDs remain unique while users edit the data.
- Performs comprehensive data validation at different stages of the editing process, such as online validation of user input, on-demand validation of non-schema enforced constraints, and validation against the XML schemas before saving data to file.

VII. Conclusions

The solution described in this paper *reduces the cost of implementing test programs* by:

- Reducing the need for manual coding for test program completion.
- Eliminating the errors that can occur during manual input of test program data.
- Using COTS software products and also allowing extension and customization.

The solution ensures *cost-effective, long-term maintainability of test programs* by:

- Storing test requirements in an industry-standard format.
- Allowing future changes to requirements while automatically preserving the consistency between requirements and implementation.
- Facilitating test results traceability by automatically maintaining references to the original test specifications.

References

[1] ATMLFramework, http://grouper.ieee.org/groups/scc20/tii/atml-family.htm, downloaded 8 July 2013

[2] Neag, I. A., Seavey, M., "Applications of IEEE P1671.1 ATML Test Description", Proc. Autotestcon 2007, pp 197 – 204, Baltimore, MD

[3] IEEE-SA Supplemental Material, web page for IEEE Std 1671.1, http://standards.ieee.org/downloads/1671/1671.1-2009/, downloaded 8 July 2013

[4] ATML Pad, http://www.atmlpad.com, downloaded 8 July 2013

[5] Jain, A., Delgado, S., "Automatic ATML test description translation to a COTS test executive", Proc. Autotestcon 2009, pp 190 - 194, Anaheim, CA