Managing the transition to IEEE 1641, via ATLAS based test systems

Ashley M B Hulme
EADS TEST ENGINEERING SERVICES (UK) Ltd
23–25 Cobham Road, Wimborne
Dorset, BH21 7PE, UK
ashley.hulme@ieee.org

***Abstract*—Over the last two years, a product has been introduced onto a Royal Air Force test system, which provides an integrated IEEE 1641 development and run-time system. The program involved the integration of several existing COTS products to provide a facility to enable the creation of a TPS using 1641 signal definitions, through to the running of the test program using established instruments and driver software. This system uses ATLAS as an intermediary between the IEEE 1641 program and the run-time system.**

There are many existing systems in the field which are based on ATLAS of various flavors. Many, if not all, of these could be upgraded to include an IEEE 1641 programming facility, which would provide compatibility with the IEEE 1761 (ATML) series of standards now being adopted by the DoD. This paper discusses how the transition to IEEE 1641 may be managed at comparatively low cost using ATLAS systems. The result of this transitional method will be a suite of validated IEEE 1641 test programs that not only run on current and legacy test systems but will provide a library of programs ready to run on future full-function 1641-based test systems.

Keywords-ATE; ATLAS; ATLAS test systems; ATS; IEEE 1641; signal modeling; test requirements;

I. INTRODUCTION

IEEE Std 1641 [1] is now well established as a standard for defining signals and tests and has become the required standard for all future test systems for the UK MoD. Over the last two years a product has been introduced onto a Royal Air Force (RAF) test system. This system, which follows a series of proof of concept exercises [3] promoted by the UK MoD over the last few years, is now fully up and running with both analog and digital test facilities [4]. This system uses ATLAS as an intermediary between IEEE 1641 and the run-time system [5].

There are many hundreds of ATLAS based systems in use throughout the US and many other parts of the world, and this provides the opportunity for IEEE 1641 to be introduced at low cost on these systems. This can be done without in any way affecting all the existing programs currently running on the system. Test programs and their associated signal definitions may then be defined using IEEE 1641, but can be supported by the existing ATLAS development and run-time systems.

II. ATLAS AND PROBLEMS OF PORTABILITY

The original intention of ATLAS was that it would provide portable test requirements that would not be system implementation dependent. Unfortunately, this did not happen in practice for several reasons. Apart from a few issues with language itself, many ATLAS implementations were not policed carefully enough and, of course, there were always time and monetary pressures that forced compromises with the way in which the language definition was interpreted on different systems.

A. Typical ATLAS portability issues

1) Interpretation: ATLAS signals (nouns and noun-modifiers) were not defined mathematically, leaving the definitions open to multiple interpretations.

2) Instrument specific uses: Many ATLAS noun modifiers were used on most systems to relate to a specific instrument resource and not used in a generic sense.

3) Inconsistent suffices: The noun modifier suffices were not defined and implemented rigidly enough, causing confusion in the meaning of some ATLAS statements.

4) Project specific adaptations: Many implementations had specific adaptations that were not adequately defined.

5) Incorrect use: Some ATLAS features were often incorrectly used.

B. Examples of ATLAS portability problems

1) Lack of clarity in some values.

An example of this is the confusion that arises when describing ripple voltages. Normally, in specifications, ripple is described as a peak to peak value. In ATLAS the noun modifier is AC-COMP, which does not take a suffix (AC-COMP-PP). But ATLAS specifies that all voltages (unless otherwise stated) are rms values. So, quite understandably, different implementations may use different interpretations of the standard.

2) Incorrect use of modifiers

This is illustrated by the Identify Event statement in ATLAS. An event is normally defined by the instantaneous value of the signal passing through a specified value. However, the statement is often written as if it is the rms value that is

expected to change. This is simply due to the –INST suffix being omitted. When porting such programs to new environments, the TPS engineer has to make a judgment—most often the correct one—but this should not be necessary.

The result of these issues is that usually ATLAS programs are only portable between test systems of the same type. If it is necessary to port the program to another (different) ATLAS system, it is these differences in interpretation that lead to errors. The real problems occur when the original intention of the test cannot be accurately determined. The TPS engineer cannot be sure that he has implemented a valid test. Another issue that may arise is when a test is implemented differently on another system, giving similar but not necessarily the correct result. Errors caused thus may never be discovered.

C. Implementation and system differences

Differences in implementation do not cause so many problems. Provided that the intention of the original test can be determined, a resolution can be found by the TPS engineer. Clearly, if a facility in one system was not available in the other, this normally becomes apparent very quickly, and steps may be taken to upgrade the target system.

D. Alternative approach

Defining a program using IEEE 1641 removes the problem of misunderstanding the test and they may still be implemented using the original underlying ATLAS system.

III. STRUCTURE OF AN IEEE 1641 PROGRAM

An IEEE 1641 program comprises a series of signal definitions for stimulus and measurement linked by the procedural part of the program defined in a carrier language.

Each different signal may be defined directly using Basic Signal Components (BSCs) or using Test Signal Framework (TSF) models. A TSF may be used in several different signals or in several instances of a signal. The carrier language used to sequence the tests will be dependent upon the specific implementation. The use of TSFs and the ability to choose the carrier language opens the door to the use of ATLAS.

The combination of the TSF library (with all the signals required for the tests) and the carrier language sequencing information form a complete definition of the test requirement. It is this information that forms the complete portable test requirement that may be implemented on any test platform, now or in the future. It is also this information that is used with the ATLAS on an existing test system.

Fig. 1 shows two segments of an IEEE 1641 test program with the signal information provided in the standard's Test Procedure Language (TPL). The TPL signal statement is enclosed within the carrier language, C# in this example. The signal detail within the TPL statement is in XML. Although this example is included to illustrate the signal information within the carrier (and it is possible to identify an 8.1 V ac signal with a frequency of 1.3 kHz), the TPS creator would not normally work directly with this code but at a higher (graphical) level.

```
// Test Action: "Apply 8 V Sinusoid"
// Test Procedure called: "05A Apply 8 V ac"
public string(10) Func_B()
{
    Console.WriteLine("Starting execution of test: Apply 8 V
        Sinusoid");
    <TPL>
    // Setup Source: "8 V Sinusoid"
    Setup
    <Signal Out='TwoWire9'  xmlns='STDBSC' xmlns:this='ATELib'
        xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
        xsi:schemaLocation='ATELib ATELib.xsd'>
      <this:acSignal name='acSignal6' voltage='{glp_IP_002}
        range 1 V to 10 V' dc_offset='' freq='{glp_IP_003}  range
        1 kHz to 2 kHz' phase_angle='' />
      <TwoWire name='TwoWire9' lo='JS-2' hi='JS-1'
        channelWidth='1' In='acSignal6'/>
    </Signal> as sigSource4;
    </TPL>
    Console.WriteLine("Finished execution of test: Apply 8 V
        Sinusoid");
    return "PASS";
}
.
.
.
    if (stepID == 32)
    {
        // Set global variable with default value
        glp_IP_002 = 8.1;
        // Set global variable with default value
        glp_IP_003 = 1.3;
        retVal = Func_B();
    }
```

Figure 1. Carrier Language with XML signals in TPL

IV. IEEE 1641 BASED TEST SYSTEM

In an IEEE 1641 based test system, or one originally designed to operate using IEEE 1641, the TSF library and associated signals would be mapped onto the system facilities directly, probably using interfaces compliant with the ATML (IEEE 1671) series of standards.

A. Using ATLAS for the implementation

ATLAS fundamentally describes signals in the same way as IEEE 1641. This is not surprising as IEEE 1641 was developed as a result of a program to update ATLAS. ATLAS includes signal constructs within the signal statements, which are similar in function to IEEE 1641 signals and includes procedural language constructs which perform the same function as the IEEE 1641 carrier language. Therefore, if the IEEE 1641 signals are translated into ATLAS signal statements and the carrier language procedural instructions are presented as ATLAS procedural statements, then ATLAS may be used to implement IEEE 1641 programs.

When using IEEE 1641 in this situation, i.e. in a system where ATLAS has been the usual programming environment, the potential problems are reduced as the IEEE 1641 development environment can be set up to map onto the ATLAS system. That does not mean that externally derived IEEE 1641 programs cannot be used, as explained later. The starting point is to consider that the system being used will only support a subset of all possible signals. This means that a

limited range of IEEE 1641 signals may be described in a TSF library created specifically for the system.

B. System TSF Library

Normally, a 1641 test program, or a suite of 1641 test programs, will come with the relevant TSF library of required signals. The alternative approach suggested here is that a system TSF library is created, providing standardized TSF models for system facilities such as power supplies, ac & dc levels, the usual signal generator functions, and rf signals, etc. These are matched to the capabilities of the system instruments. These predefined TSF models may then used by the test programs by providing the appropriate values for the TSF attributes. This method ensures that the test programs are compatible with the ATE capabilities.

C. Creating the Test Programs

IEEE 1641 test programs may then be created referencing the predefined TSF models and using an appropriate carrier language. In practice, this is best done using a language independent tool such that signals are linked within the appropriate structure and the carrier language is created automatically, removing one layer of work for the programmer. After completing this process, there is a test program with IEEE 1641 signals (usually in XML) and a carrier language (for example, C#). To use these on the existing ATLAS system now requires a translation process. This can be completed mechanistically using an automatic tool.

D. Translation to ATLAS

Converting the IEEE 1641 program to ATLAS is a two part process.

1) Sequence

The carrier language program sequential information is converted into ATLAS control statements. This is where the aforementioned tool is helpful, in that it produces regular and defined carrier language structures, simplifying the translation process.

2) Signals

Each TSF model in the library will have an equivalent ATLAS signal statement structure, which is inserted into the ATLAS program sequence structures created from the carrier language.

Of course, it is not going to be as simple as that. There are always going to be signal features that are easy to express in IEEE 1641 that do not fit into standard ATLAS. These may be dealt with in several ways, but the most convenient method is to create some special ATLAS to cover the problem. It is not only possible to add nouns and noun-modifiers (as allowed in standard ATLAS) but possible to change the ATLAS more liberally, which would normally be outside the scope of ATLAS. The justification for this is the fact that the ATLAS is used here as a means to an end, and will never exist outside of this environment.

Fig. 2 shows the segment of ATLAS generated to apply the 8.1 V ac signal described in Fig. 1. Note that the ATLAS is not true ATLAS, e.g. lower case is used where it should be upper case. This is not important provided that it compiles and runs correctly because the ATLAS will not need to ported to any other environments.

```
   DEFINE, 'DP_BFunc_B', PROCEDURE
      RESULT
        ('_this_' IS 'DP_B',
         '_return_' IS STRING(10) OF CHAR) $
        OUTPUT, C'Starting execution of test:
Apply 8 V Sinusoid' $
C sigSource4.acSignal6.acSignal: $
      SETUP, AC SIGNAL,
        VOLTAGE 'glp_IP_002' range 1 V to 10 V,
        FREQ 'glp_IP_003' range  1 kHz to 2 kHz,
        CNX HI JS-1  LO JS-2  $
C sigSource4.acSignal6.acSignal: $
      CONNECT, AC SIGNAL,
        VOLTAGE 'glp_IP_002' range 1 V to 10 V,
        FREQ 'glp_IP_003' range  1 kHz to 2 kHz,
        CNX HI JS-1  LO JS-2  $
      OUTPUT, C'Finished execution of test:
Apply 8 V Sinusoid' $
C the beginning of the assignment section $
      CALCULATE, '_return_' = C'PASS' $
C the end of the assignment section $
C beginning of the return section $
      LEAVE, 'DP_BFunc_B' $
C the end of the return section $
   END, 'DP_BFunc_B' $
.
.
.
C the beginning of the assignment section $
   CALCULATE, 'glp_IP_002' = 8.1 $
C the end of the assignment section $
C the beginning of the assignment section $
   CALCULATE, 'glp_IP_003' = 1.3 $
C the end of the assignment section $
```

Figure 2. Example of ATLAS created by a translator

E. Debugging and Running the Program

The ATLAS created by the translator is then processed by the ATLAS toolset resident on the system and is run using the normal ATLAS run-time executive. In the early days of a new implementation, when the first few programs are commissioned, there may be some translation (or mis-translation) issues to solve, but this is no different than when any other software suite is implemented on a system for the first time. After all the basic translations have been proven, any further issues mainly involve new TSF models and their ATLAS equivalents.

All the existing device drivers are still valid, although some may need upgrading and some new drivers may be required to cover signal enhancements.

The new programs will also be compatible with all the existing system support functions such as results logging, and operator familiarity with the system is not affected.

F. Simple overview

Fig 3. provides a simplified schematic of the processes involved. The IEEE 1641 program information is translated into ATLAS using the Signal equivalence file, which contains enough information to represent all the signals in the system TSF library.

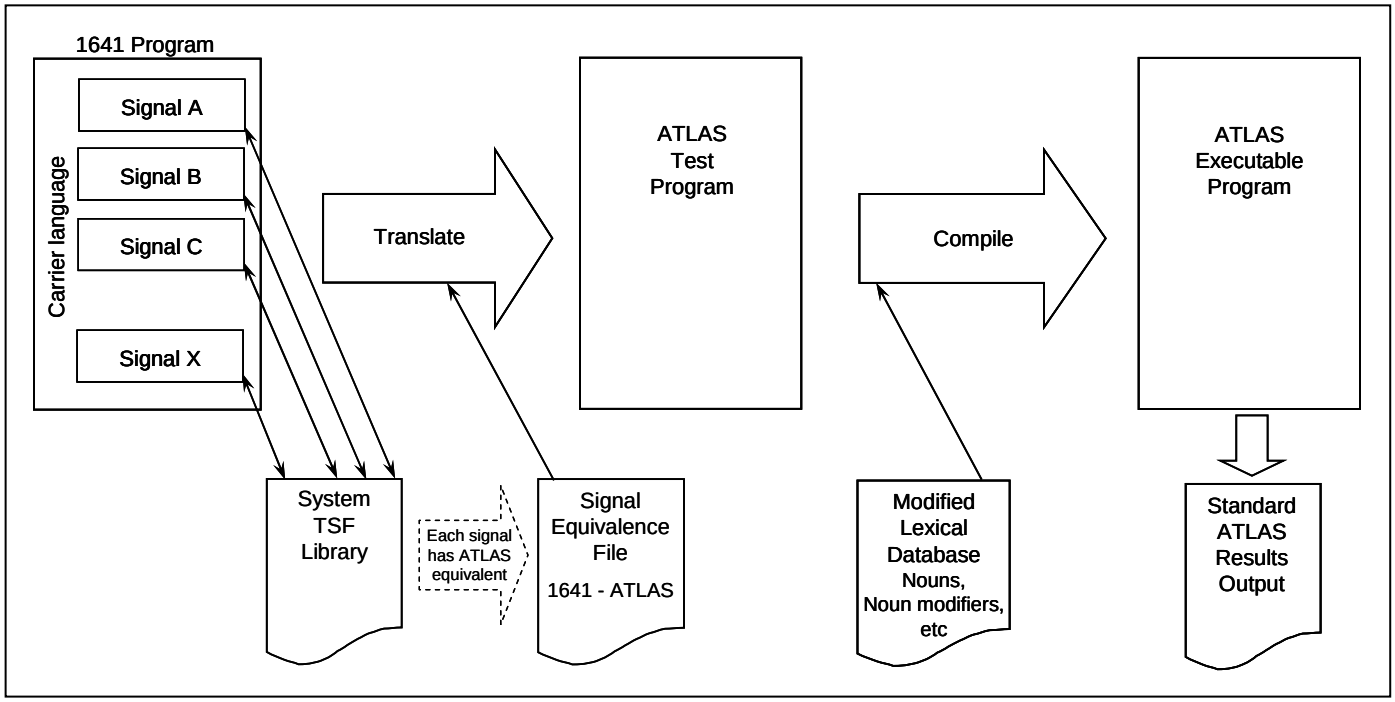

Figure 3. Simplified diagram of the IEEE 1641 to ATLAS translation and compilation process

The ATLAS compilation process is as standard but with the addition of any special lexical data for newly introduced signals.

G. Imported Programs

It is also possible to implement 1641 programs not originally designed for this system. Of course, the system must be capable of hosting the programs with system enhancements if necessary. These programs will arrive with their own TSF library, which may not be a perfect match for the existing system TSF library and the signal translation file. The solution for this is to create additional parallel TSFs which align with existing system facilities but consume the new signals' TSF attribute data. The imported program information is preserved intact but now may also run on the ATLAS based system.

V. A Practical System

Such a system has been implemented in the UK on an RAF system at DSG, Sealand [4]. It uses SigBase (from the TYX Corporation) as the IEEE 1641 Integrated Development Environment complete with its ATLAS translation system together with newWaveX-SD software (from EADS) to define the signals. It was integrated by EADS Test Engineering Services in the UK, who also configured the signal translation files to ensure that the appropriate ATLAS was created for the 1641 defined signals. Fig. 4 shows a typical segment from the translation file, which causes the IEEE 1641 signal information for the TSF "acSignal" to be translated into the appropriate ATLAS signal statement fields.

The examples of code shown in Fig. 1, Fig. 2 and Fig. 4 are all "under the hood" in the system as implemented. The program is designed graphically in SigBase via a flow-chart based input mechanism (see Fig. 5). SigBase also controls the translation process and interfaces with the run-time system. The signals are also created graphically using newWaveX-SD as shown in Fig. 6. The newWaveX package will also simulate the signals to enable the TPS creator to ensure that the signal as defined is correct.

```
<acSignal noun="AC SIGNAL">
  <attribute name="ac_ampl">
    <qualifier name="trms">VOLTAGE-TRMS</qualifier>
    <qualifier name="pk_pk">VOLTAGE-PP</qualifier>
    <qualifier name="pk">VOLTAGE-P</qualifier>
    <qualifier name="pk_pos">VOLTAGE-P</qualifier>
    <qualifier name="pk_neg">VOLTAGE-P-NEG</qualifier>
    <qualifier name="av">VOLTAGE-AV</qualifier>
    <qualifier name="inst">VOLTAGE-INST</qualifier>
    <qualifier name="inst_max">VOLTAGE-P-POS</qualifier>
    <qualifier name="inst_min">VOLTAGE-P-NEG</qualifier>
    <qualifier name="default">VOLTAGE</qualifier>
  </attribute>
  <attribute name="dc_offset" default="0 V">
    <qualifier name="default">DC-OFFSET</qualifier>
  </attribute>
  <attribute name="freq">
    <qualifier name="default">FREQ</qualifier>
  </attribute>
  <attribute name="phase" default="0 rad">
    <qualifier name="default">PHASE-ANGLE</qualifier>
  </attribute>
</acSignal>
```

Figure 4. Segment of ATLAS translation file

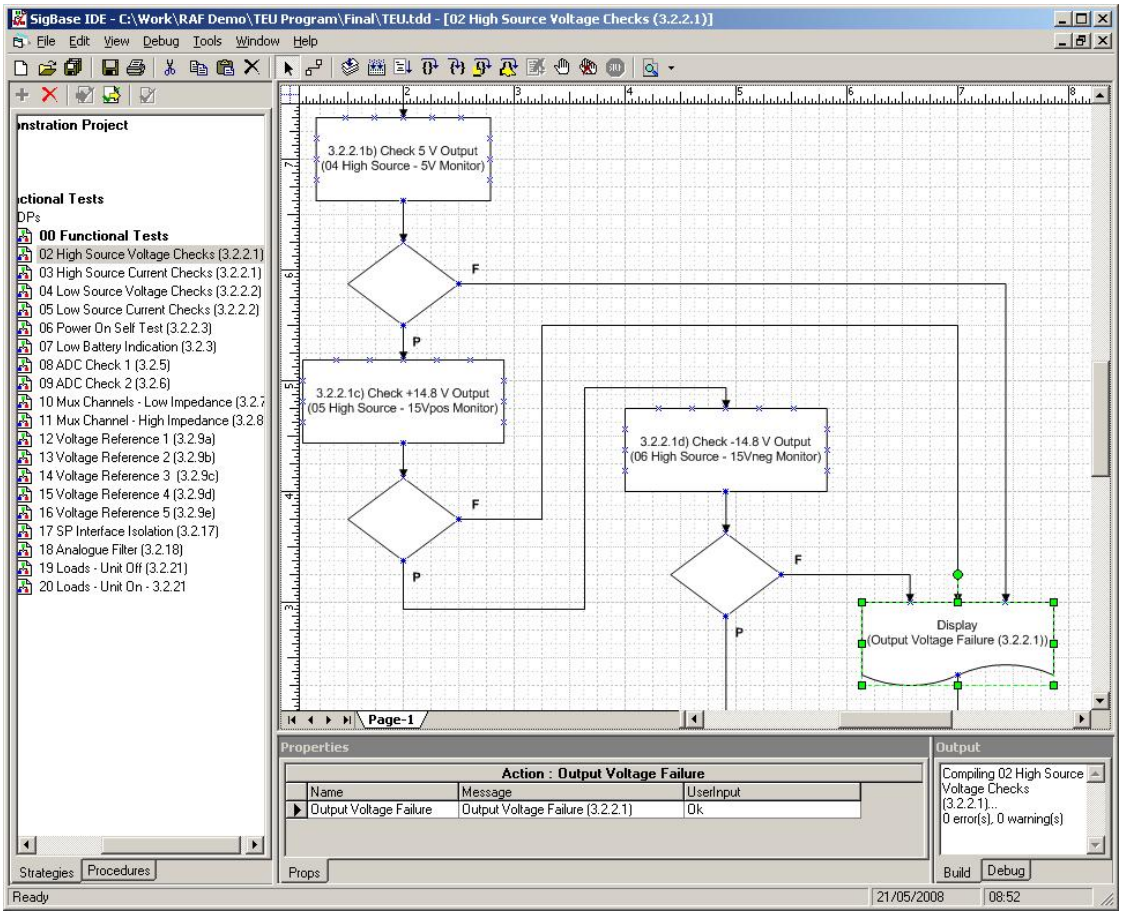

Figure 5. Test Program generation using a flow-charting process

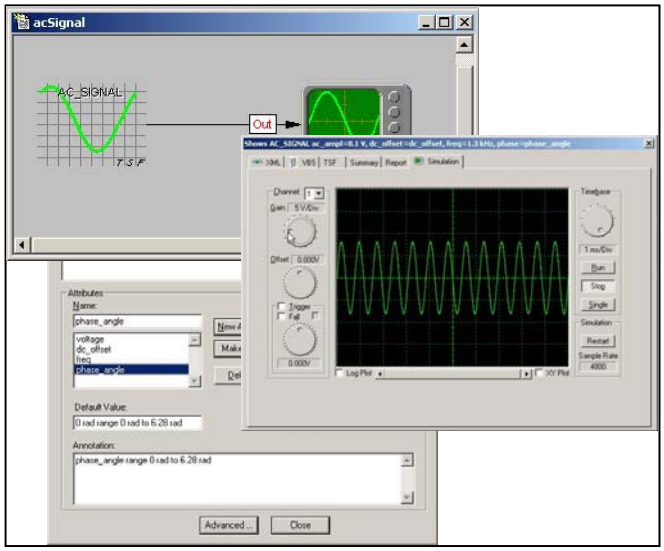

Figure 6. Signal definition and simulation

VI. SUMMARY

This method of hosting IEEE 1641 programs using an existing ATLAS compiler and run-time suite represents an excellent way of getting IEEE 1641 programs onto an existing test system. The obvious result is that IEEE 1641 based programs are able to be created and run on a standard ATLAS test system as is used in many military test environments. The benefit is that there is also a suite of IEEE 1641 programs that can be ported onto other IEEE 1641 test systems. This applies to new purely IEEE 1641 based systems and other ATLAS based systems with different ATLAS subsets and implementations.

The key point being that it is the IEEE 1641 programs with their signal descriptions supported by mathematical definitions that are ported, not the implementation specific underlying ATLAS programs.

REFERENCES

[1] IEEE Std 716™–1995, IEEE Standard Automatic Test Language for All Systems. Institute of Electrical and Electronics Engineers, Inc.

[2] IEEE Std 1641™–2004, IEEE Standard for Signal and Test Definition. Institute of Electrical and Electronics Engineers, Inc.

[3] M. Brown, K. Ellis, and A. Hulme, State of IEEE 1641Standard, Applications and UK MOD Policy, IEEE AUTOTESTCON 2008 Proceedings, pp 295–300

[4] A. Hulme and K. Nash, Implementing IEEE 1641 – Using a complete system, IEEE AUTOTESTCON 2008 Proceedings, pp301–307

[5] C Grard, A practical use of the 1641 signal definition using the Test Procedure Language and a Carrier Language, IEEE AUTOTESTCON 2008 Proceedings, pp 314–318