Best Practices for Describing Digital Serial Buses and Bus Test Operations Using IEEE 1671 ATML and IEEE 1641 Signal and Test Definition

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

Chris Gorringe
Spherea Test & Services
Christchurch, UK
chris.gorringe@spherea.co.uk

***Abstract*— The recently published revisions of IEEE Std 1671.3 ATML UUT Description and IEEE Std 1671.1 ATML Test Description contain many new features that support the description of serial bus testing. This paper proposes a set of best practices for using these new features to describe UUT serial buses and bus test operations. The application of these best practices produces UUT and test descriptions that are simple, accurate, and maintainable for the lifetime of the UUT.**

Keywords—IEEE 1671, ATML, IEEE 1641, serial bus, testing

I. INTRODUCTION

IEEE 1671.3 ATML UUT Description specifies a standard XML-based format for describing the characteristics of a Unit Under Test (UUT) [1].

IEEE 1671.1 ATML Test Description specifies a standard XML-based format for describing UUT test requirements [2]. This format improves on the legacy Test Requirements Documents (TRD) formats by allowing the detailed description of test behavior through Operations using signal definitions conformant to IEEE 1641 [3]. This method provides sufficient detail to allow the automatic generation of test programs from requirements [4] [5].

The use of ATML Test Description to describe test requirements for UUTs with digital serial buses addresses traditional challenges:

- Ensure support for legacy, modern, and future buses
- Separate bus configuration operations from bus exchange operations
- Describe complex data encoding in bus payloads
- Describe periodic message transmission (frames)

The 2017 revision of the IEEE 1671.1 and IEEE 1671.3 standards addresses these challenges by adding specialized descriptive elements for bus messages and frames and specialized Operation types for bus exchanges.

In this paper we propose a set of best practices for using the revised standards to develop test requirements for UUTs with digital serial buses.

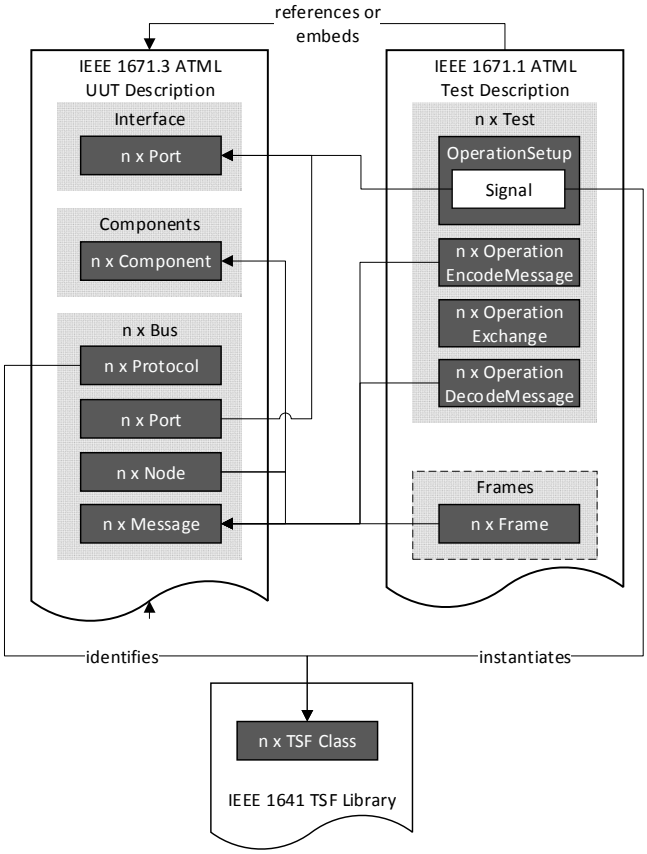

Fig. 1. Standard XML documents used to UUT describe test requirements

II. DEVELOPMENT PROCESS

Fig. 1 shows the standard XML documents used to describe test requirements for a UUT, along with their relationships. In this figure the elements specific to digital serial buses are shown with a dark background. The dependencies between XML documents are as follows:

- The **ATML Test Description** instance document describes test requirements. It either references a standalone **ATML UUT Description** instance document, which describes the UUT being tested, or

embeds XML content conformant to the ATML UUT Description specification.

- The bus Protocols specified in ATML UUT Description identify the TSF Classes that define the protocol. These classes are part of an **IEEE 1641 TSF Library**.
- The bus Setup Operations that appear in the ATML Test Description document instantiate TSF Classes from the IEEE 1641 TSF Library. The bus Exchange Operations use the signal instances to describe the data exchanges required for testing the UUT.

The typical process for developing test requirements for UUTs with digital serial buses includes the following steps:

1. Reuse, adapt or develop an IEEE 1641 TSF Library with signal definitions for serial buses, identifying their key interface properties:
   1.1. Define the serial bus protocols supported by the UUT through TSF classes
2. Describe UUT serial buses in the ATML UUT Description instance document:
   2.1. Describe the physical buses through *Component*, *Node*, and *Port* elements
   2.2. Describe the protocols supported by the serial buses through *Protocol* elements
   2.3. Describe the format of bus payloads through *Message* elements
3. Describe UUT test requirements in the **ATML** Test Description instance document:
   3.1. Describe the repetitive data exchanges required for testing the UUT through *Frame* elements (where applicable)
   3.2. Describe bus-related test behavior through *Operations*

NOTE—Each of these steps produces content for XML instance documents and XML schema documents. While this paper references various XML elements and attributes, it does not imply that the XML must be edited manually, as text. Given the complexity of the data formats involved, specialized software development tools or editors should be used.

III. Getting Started

For background information and examples relevant to the topic of this paper, we recommend the following resources:

- IEEE 1641.1-2013 [6], Clause 10, "Test Signal Frameworks"
- IEEE 1641.1a-2018 [7], Amendment 1, "Addition of Guidelines for Producing Reusable Test Signal Frameworks for Use on Platforms Utilizing Automatic Test Markup Language"
- IEEE 1671.3-2017 [1], Annex C, "Describing UUT Serial Digital Buses"
- IEEE 1671.1-2017 [2], Annex D, "Describing serial digital bus exchanges"
- Example TSF library and XML instance documents from the IEEE 1671.1-2017 and 1671.3-2017 XML packages, available for download at http://standards.ieee.org/downloads/

IV. Defining IEEE 1641 TSF Classes for Digital Serial Buses

A. Introduction to TSF Models and Interfaces

To support the specification of test requirements in ATML, each serial bus protocol supported by the UUT must be defined in a TSF class. A TSF class has two parts:

1. The *model definition*, describing the operation of the bus protocol through a combination of standard Basic Signal Components (BSCs) and possibly other TSFs.
2. The *interface definition*, describing the attributes of the protocol that are configurable, or are used to describe a capability of an ATE resource.

The model can be omitted for buses defined by industry standards such as ARINC 429 and Mil-Std-1553B. A model must be provided for all non-standard bus protocols and when multiple standard protocols are combined in a custom stack, for example FTP-TCP-IPv4-IEEE 802.3. In this, each standard layer is represented in the model by a TSF, referencing the corresponding standard.

TSF classes are packaged in TSF Libraries. The following files are typically generated when creating a TSF library:

1. **XML file (.xml)**. The file contains one <TSF> section for each TSF class. This section has two main parts, <model>, with the *model definition* and <interface>, with the *data interface definition*.
2. **XML Schema file (.xsd)**. This file contains the *data interface definitions* for all TSF classes in the library. Note that this file is only provided for convenience (i.e., to support schema validation); the information it contains is redundant, being already included in the XML file.
3. **Interface Definition Language (IDL) file (.idl)**. This file contains the *programmatic interface definitions* for software components implementing the TSF classes.

In the following we will focus on the definition of TSF class interfaces. The development of models for non-standard buses and for protocol stacks [8] is outside the scope of this paper.

B. Reusable TFS Libraries

Besides describing UUT test requirements, TSF Libraries are also used to describe ATE hardware capabilities, being referenced from IEEE 1671.2 ATML Instrument Description and IEEE 1671.5 ATML Test Station Description.

1. If TSF Libraries describing the serial bus protocols already exist (for example, to describe the capabilities of a target ATE), then reuse those libraries when defining the UUT test requirements.

1.1. If the capabilities described in some TSF class are insufficient for testing the UUT (for example, a higher bit rate is required), modify the TSF class as needed. For non-breaking changes (i.e., all signal definitions that validate against the original version will also validate against the new version), increment the 'version' attribute of the root element. For breaking changes, modify the namespace of the TSF Library.

2. If necessary, develop TSF Libraries for the serial bus protocols of the UUT.

2.1. To maximize the opportunities of targeting various ATE platforms in the future, you should only define the protocols and protocol features that are required for testing the UUT, which increases their reuse.

2.2. If you are developing test requirements for a family of UUTs, then the TSF Library you are creating should be shared by all the ATML Test Description and ATML UUT Description documents for these UUTs. The TSF Library will define the envelope of requirements across all target UUTs.

2.3. For convenience, you can separate TSF class definitions into multiple TSF libraries, for example a library for digital serial buses, one for digital parallel signals, one for low-frequency analog signals, and so on.

C. *Developing a TSF Class Definition*

Most digital serial buses implement a layered protocol stack [8]. While the number of layers and their role are different from bus to bus, the bottom layer is always the *physical layer*, representing the physical transmission lines and associated devices such as transformers, hubs, etc., while the top layer is the *application layer*, representing the application programming interface (API) where the bus interface software interacts with the test software, exchanging *data payloads*.

Note that multiple protocols or protocol stacks can be supported by a given physical bus. For example, a wired Ethernet bus could be used to upload firmware via FTP and to retrieve BIT data via HTTP. In this situation a separate TSF class will be needed for each protocol stack (FTP / TCP / IPv4 / IEEE 802.3 and HTTP / TCP / IPv4 / IEEE 802.3).

1) *Defining a Serial Bus Protocol*

To create a TSF class for a protocol or a protocol stack, perform the following steps:

1. Identify all serial bus protocols used in the operation and testing of the UUT.

2. For each protocol or protocol stack, create a TSF class. Assign a descriptive class name, matching the nomenclature of the applicable bus standard. Use camel case capitalization or, if emulating a legacy ATLAS subset, use uppercase with '_' separators.

3. For standard protocols, determine the applicable standard document and reference it using the *model/standard* element. Note that this element can only reference a single standard. To reference multiple standard documents, for example for a protocol stack, or to provide more extensive referencing information, for example, document locations on the World Wide Web, use the complementary referencing mechanism supported by the *ProtocolDescription* element from the ATML UUT Description schema.

RS-232: If the legacy ATLAS syntax defines the BUS PROTOCOL "RS-232", then the corresponding TSF class is named "RS_232".

RS-232: To provide an exact reference to standard "TIA-232, Revision F, October 1, 1997", the *ProtocolDescription* element has a child element of type *Identification*, with the value "TIA-232" and the attributes version="Revision F" and date="2012-12-01".

2) *Defining the Configurable Protocol Parameters*

Define the configurable parameters of serial bus protocols through attributes of the corresponding TSF class, as follows:

1. For each protocol, identify the configurable protocol parameters, for example:

RS-232: voltage levels, baud rate, parity, number of data bits, start bits, and stop bits

ARINC429: bit rate (can be either 100 Kb/sec or 12.5 Kb/sec)

Mil-Std-1553B (counter-example): because the transmission bit rate is fixed, equal to 1.0 Mb/sec, it is not a configurable protocol parameter

2. For each configurable parameter identified above, create an attribute in the corresponding TSF.

2.1. Assign a descriptive attribute name, matching the nomenclature of the applicable bus standard. Use the recommended capitalization for IEEE 1641 signal attributes (lowercase and '_' word separator).

RS-232: If the legacy ATLAS syntax defines the noun modifier "WORD-LENGTH", then the corresponding TSF class attribute is named "word_length".

2.2. Assign to the attribute one of the data types supported in IEEE 1641. Table I provides guidelines for data type selection in several typical situations.

2.3. Where mandated by the protocol specification, provide a range for the Physical attribute value.

RS-232: In the TSF class "RS232" described in Annex C of IEEE 1671.3-2017 the attributes 'level0' and 'level1' have a range of -12.0 V to 12.0 V.

2.4. Unless specified otherwise, all TSF class attributes are optional. Leave as "optional" the attributes for which a default value can be assigned. Assign a default value, either as a constant or as an expression describing a dependency on the other attributes.

ARINC429: In the TSF class "Arinc429" described in Annex C of IEEE 1671.3-2017 the attribute 'bit_rate' is optional; its default value is 12.5 kHz.

TABLE I. IEEE 1641 DATA TYPES FOR PROTOCOL PARAMETERS

Parameter Type	IEEE 1641 Data Type	Example
Physical quantity representing the signal on the transmission lines	One of the Physical types specified in Clause A.3 of IEEE 1641-2010	**RS-232** voltage levels: Voltage
Dimensionless numeric quantity, discrete values	Restriction of int, long, or double, as appropriate	**RS-232** baud rate: data type int, restricted to the enumeration values 75, 110, 134, 150, …
Non-numeric, enumerated values	Restriction of string. Assign descriptive names, matching the nomenclature of the bus standard. Use the '$' prefix, to avoid these values being mistaken for expressions.	**RS-232** parity: $Even, $Odd, $None, …

2.5. Mark as "required" all attributes for which a default value cannot be assigned. Attribute values will have to be specified in every test requirement and capability description.

3) Defining the Data Payloads

Define the data payloads of serial bus protocols through attributes of the corresponding TSF class, as follows:

1. For each protocol, identify and characterize the data payload, for example:

   RS-232: string, or array of bytes (depending on the application)

   ARINC429: 32-bit word

   Mil-Std-1553B: 20-bit word, multiple words per message

2. For each data payload identified above, create one or more attributes in the corresponding TSF class, following these guidelines:

   2.1. To simplify the test program, define the API in a manner in which the construction of the data word (including any error detection or correction) can be delegated to the upper layer of the protocol stack. In addition, you may want to provide attributes for the overall data word or for its calculated parts. This allows the test description to specify incorrect data words, for example to perform fault injection.

   ARINC429: The interface of the TSF class "Arinc429" described in Annex C of IEEE 1671.3-2017 defines separate attributes for the components of the data word (label, SDI, message, SSM). The test description (and implicitly the test program) specifies the values of these parts. The implementation of the resource is responsible for combining the parts in a 32-bit word. The attribute 'data' provides access to the constructed data word.

   Furthermore, the interface of the "Arinc429" TSF class defines the attribute 'parity' with the possible values $Odd, $Even, and $None. The test description (and implicitly the test program) specifies the desired type of parity. The implementation of the resource is responsible for calculating the party bit. The attribute '_parity' provides access to the calculated parity bit. The test description can specify for this attribute an incorrect value, to verify the behavior of the UUT in the presence of incorrect parity data (fault injection).

   2.2. To maximize the potential for reuse, define the TSF class interface in a generic manner, which reflects the bus interface standard and is independent of a specific application or UUT.

   ARINC429: Individual UUTs encode various classes of avionics data (airspeed, altitude, etc.) in parts of the data message. However, the TSF class interface should define a general-purpose 19-bit data "packet" and not these avionics data classes. The encoding of avionics data classes into ARINC 429 messages should be described in the ATML UUT Description document through *Message* elements, as discussed later in this paper.

   2.3. Assign to each attribute one of the data types supported in IEEE 1641. Table II provides guidelines for data type selection in several typical situations.

TABLE II. IEEE 1641 DATA TYPES FOR PROTOCOL PAYLOADS

Payload Type	IEEE 1641 Data Type	Example
String	string	**RS-232** with string messages: string
Binary, fixed length	byte, short, int, or long, depending on the word length	**ARINC429**: int
Binary, variable length	array of int or long, depending on the word length	**RS-232** with binary data: array of byte **Mil-Std-1553B**: array of int

   2.4. Leave as "optional" all the attributes for which a default value can be assigned. Assign a default value, either as a constant or as an expression describing a dependency on the other attributes. Mark as "required" all attributes for which a default value is not assigned.

   ARINC429: In the TSF class "Arinc429" described in Annex C of IEEE 1671.3-2017 the attribute 'data' is optional; its default value is specified through an expression that combines the values of other attributes.

V. DESCRIBING UUT DIGITAL SERIAL BUSES

The physical implementation and the operation of UUT serial buses is described in the ATML UUT Description document. Refer to Figure C.1 in IEEE 1641.3-2017 for the

relationship between various XML elements used for this purpose.

NOTE—While in principle the ATML UUT Description should describe the operation of the UUT independent of any testing requirements, in practice it is often necessary to take into account test requirements when developing the ATML UUT Description. For example, a serial bus model used in testing might need to support out-of-nominal ranges for some protocol parameters, to facilitate fault insertion.

A. Item Descriptions vs. Item Instances

The physical implementation of buses is described through *Protocol*, *Node*, and *Port* elements, while the payloads are described through *Message* elements. Note that the characteristics of protocols, ports, and messages are not described through descendants of these elements. Instead, separate *ProtocolDescription*, *NodeDescription*, and *MessageDescription* elements must be created and referenced by ID from *Protocol*, *Node*, and *Port*. This technique allows items with identical or very similar characteristics to share a common description, thus reducing the size and complexity of the ATML documents.

B. Introduction to Attributes

As illustrated in Fig. 1, UUT test requirements are described in the ATML Test Description document through *Operations* that assign values to signal attributes representing configurable protocol parameters. Many protocol parameters have fixed values, which do not change during UUT operation or testing, or could change occasionally to perform fault insertion. To avoid repeating identical values in multiple *Operations*, the constant or common values of protocol parameters can be specified in the ATML UUT Description document, through *Attribute* elements assigned to bus, node, or message elements. Select the location of each *Attribute* according to the scope of value to be specified, as follows:

1. Specify protocol parameter values common to the entire bus using *Attribute* elements under *Bus*.
2. Specify protocol parameter values specific to a bus node using *Attribute* elements under *Node* or *NodeDescription*.
3. Specify protocol parameter values specific to a message using *Attribute* elements under *MessageDescription*.

C. Introduction to Messages

Avionics buses are frequently used to exchange data representing values of physical quantities, such as set points for actuators, measured values from sensors, or calculated values exchanged between computers. Because the data payloads supported by most serial bus protocols are designed to hold arbitrary content (digital data words, strings, data streams), each application must define a format for packaging the values to be exchanged into the available payload space. Packaging is especially important for older buses, with low data rates and size-constrained payloads.

As illustrated in Fig. 1, UUT test requirements are described in the ATML Test Description document through *Operations* that perform bus exchanges by writing the data to be transmitted to signal attributes representing data payloads and by reading the received data from attributes. The values to be transmitted must be encoded (packaged) before being assigned to the attribute. Similarly, the data read from the attribute must be decoded (unpackaged) into values after reception. Specifying the encoding/decoding formats in the ATML Test Description document, in conjunction with each serial bus *Operation*, would greatly complicate the test description. Instead, the applicable encoding/decoding formats are defined in the ATML UUT Description document and are simply referenced from *OperationEncodeMessage* and *OperationDecodeMessage*.

For detailed information on defining and using messages refer to Annex C of IEEE 1671.3-2017 and Annex D of IEEE 1671.1-2017. For representative examples of message encoding for the ARINC 429 and Mil-Std-1553B buses, refer to the example XML instance documents from the IEEE 1671.1-2017 and 1671.3-2017 XML packages.

D. Describing Buses

Describe each digital serial bus of the UUT as follows:

1. Create one *Bus* element for each digital serial bus of the UUT.
2. Use *Bus/Attributes/Attribute* elements to specify the values of protocol parameters common to the entire bus.

   ARINC429: ARINC 429 buses can have a bit rate of either 100 kb/sec or 12.5 kb/sec. Because all the nodes connected to a bus share the same rate, the bit rate can be specified using an *Attribute* element located under *Bus*.

E. Describing Bus Protocols

Describe the protocols used by the UUT serial buses as follows:

1. Create one *ProtocolDescription* element to describe each unique protocol or protocol stack supported by the UUT. For standard protocols, assign to this element the type *StandardProtocolDescription*, referencing the applicable standard specifications and the previously created TSF class that defines the protocol or protocol stack.
2. Create one *Bus/Protocols/Protocol* element for each protocol supported by each bus. For each *Protocol*, reference the applicable *ProtocolDescription*. Note that multiple protocols or protocol stacks can be supported on a single bus.

   Ethernet: For the Ethernet example introduced previously in Section III C, *Protocol* elements will reference two *ProtocolDescription* elements describing the protocol stacks FTP-TCP-IPv4-IEEE 802.3 and HTTP-TCP-IPv4-IEEE 802.3.

F. Describing Bus Nodes

Describe the nodes (terminals) of the UUT serial buses as follows:

1. Create one *NodeDescription* element for each node type with unique characteristics. Use *Attribute* elements to describe the characteristics common to all nodes of this type.

2. Create one *Bus/Nodes/Node* element for each bus node (terminal) located within the UUT. For each *Node*, reference the applicable *NodeDescription*. Use *Attribute* elements to describe the characteristics unique to each node.

   RS-232: Serial buses are often used to upload firmware and to command the UUT during testing. In these cases, the UUT contains one node (the other node being the bus test instrument). Although a single UUT node exists, due to the XML schema structure you still have to create a *Node* element and a *NodeDescription* element.

   Ethernet: A Line Replaceable Unit (LRU) contains multiple identical cards that digitize sensor data. Each card has an Ethernet interface with a unique IP address. The card interface ports are connected to the external Ethernet port of the LRU through an Ethernet switch. The cards understand the same commands and return identically structured data payloads. In this case, each card is represented by a *Node* element and all the *Nodes* reference a single *NodeDescription*. The messages that can be transmitted or received by all the cards are described through *Message* elements under *NodeDescription*. An *Attribute* under each *Node* contains the unique IP address of the corresponding card.

G. Describing Bus Messages

Describe the messages exchanged through the UUT serial buses as follows:

1. Create one *MessageDescription* element to describe each unique encoding/decoding format supported by the UUT buses. Use *Attribute* elements to specify parameters unique to the message description.
2. Create one *NodeDescription/Messages/Message* element for each encoding/decoding format supported by the corresponding node type.

   ARINC429: The ARINC429 specification defines the encoding formats for "parameters" commonly used in avionics systems such as altitude, airspeed, orientation, and so on. Many of these parameters are being transmitted periodically during flight, emulating an analog signal. The required transmit rate depends on the rate of change of the underlying physical signal and on accuracy requirements. Because the ARINC429 document does not specify transmit rates, only acceptable ranges, the transmit rate to be used in conjunction with a particular UUT must be specified, using an *Attribute* element located under *MessageDescription*.

VI. Describing Test Requirements for UUT Serial Buses

A. Testing with the Bus vs. Testing of the Bus

The term "bus testing" is commonly used with two distinct connotations:

1. Using a bus to exchange data with the UUT while performing some other type of tests on the UUT. Having the tests pass is considered sufficient proof that the bus operates correctly.
2. Explicitly testing the bus interface components and ports within the UUT. These tests can include:
   - Verifying the UUT response to nominal bus traffic, possibly at different points inside the nominal performance envelope.
   - Verifying the UUT response to out-of-nominal bus traffic from the ATE (incorrect voltage levels, bit rate, parity bit, etc.).
   - Verifying the quality of service for the bus traffic (for example, the percentage of collisions under heavy traffic).

While the same modeling elements are used in both scenarios, the explicit testing of the bus often requires access to additional bus control parameters and the ability to set parameters to out-of-nominal values.

RS-232: When the bus is used for exchanging data with the UUT during test, the test program configures the voltage levels used by the bus test instrument to the nominal values expected by the UUT, for example +/- 12 V. When explicitly testing the bus, the test program could set the voltage levels to lower or higher values, to verify that the UUT operates correctly within some voltage range.

ARINC429: In this example we use the "Arinc429" TSF class from the example TSF library included in the IEEE 1671.1-2017 and 1671.3-2017 XML packages. When the bus is used for exchanging data with the UUT during test, the test program configures the bus test resource by setting the 'parity' signal attribute to $None, $Odd, or $Even, to match the parity type expected by the UUT. For each data transmission, the bus test resource calculates the value of the parity bit, represented by the signal attribute '_parity', and uses the calculated value in the transmission. When explicitly testing the bus, the test program could set the signal attribute '_parity' directly, to a value that does match the parity type expected by the UUT. The test program could then verify that the UUT detects the parity error and responds as expected.

B. Introduction to Bus Test Operations

The ATML Test Description schema contains a specialized Operation for describing basic serial bus exchanges, *OperationExchange*. To describe protocol configuration and data transfers in a bidirectional exchange, use this *Operation* in conjunction with *OperationSetup* and *OperationFetch* as illustrated in Fig. 2.

RS-232: The ATE sends a string command to the UUT, through an RS-232 bus, requesting it to perform Built In test (BIT). The UUT responds with a numeric BIT code. The *Operations* describing this exchange perform the following functions:

1. *OperationSetup* defines a signal by instantiating the "RS232" TSF class, then configures protocol parameters by settings attributes of the newly defined signal for baud rate, parity, data bits, and so on. A reference to the signal is stored in a variable, for use by subsequent *Operations*.

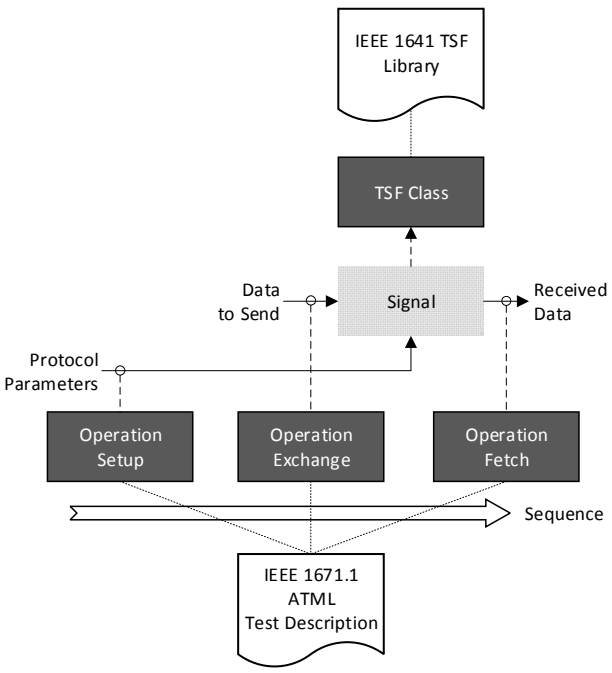

Fig. 3. Describing a bidirectional bus exchange using *Operations* and related elements

The signal defined by the *Operation*, shown in Fig. 3, has the following components:

a. *tdtsf:RS232*: instance of the TSF class, represents the digital bus signal. Its attributes represent the configurable protocol parameters and the data payload.

b. *std:Measure*: standard Basic Signal Component, represents the retrieval of received data after the exchange. The signal attribute named 'attribute' identifies the TSF class attribute representing the payload (i.e., "what to measure"); the value of this attribute must be "data". The signal attribute 'measurement' holds the received data at the end of the exchange.

c. *std:ToWire*: standard Basic Signal Components, represent the connections of serial bus data lines to UUT pins.

2. *OperationExchange* retrieves the data to send from a hardcoded value, a parameter or a variable and assigns it to the signal attribute 'data' representing the "transmit" data payload, triggering a bus exchange. The bus test instrument sends the data to the UUT, which sends back response data. The instrument records the received data in memory, making it available for retrieval.

3. *OperationFetch* reads the response data through the signal attribute 'measurement', representing the "receive" data payload and assigns it to a variable or a test result.

For detailed information on using the bus test *Operations* refer to Annex D of IEEE 1671.1-2017. For a representative example of describing a simple data exchange through the RS-232 bus, refer to the example XML instance document "Rs232Simple.xml" from the IEEE 1671.1-2017 XML package.

C. Using Bus Test Operations with Message Encoding

In many test procedures the values sent through serial buses must be converted from their native formats (boolean, byte, integer, float, etc.) to the format of the bus protocol payload (encoded). Similarly, the received payload data must be converted back to native formats (decoded). The ATML Test Description schema contains two specialized *Operations* for encoding and decoding, *OperationEncodeMessage* and *OperationDecodeMessage*. These *Operations* make use of message descriptions provided in the ATML UUT Description document, as discussed earlier in the paper.

RS-232: The ATE sends a string command to the UUT, through an RS-232 bus, requesting it to perform a measurement on a specific channel. The command string contains the channel number. The UUT responds with a string that contains the measured value in floating-point format. The *Operations* describing this exchange, shown in Fig. 4, perform the following functions:

1. *OperationSetup* operates identically to that in the previous example.

2. *OperationEncodeMessage* retrieves the channel number from a test parameter or a variable, encodes it in string format according to a *MessageDescription*, and stores the result in a *LocalVariable*.

3. *OperationExchange* retrieves the encoded string from the *LocalVariable* and then operates identically to that in the previous example.

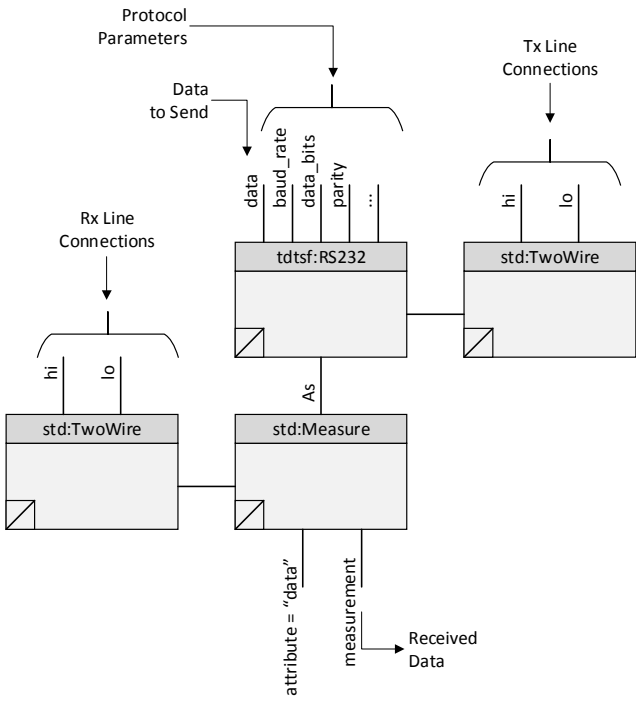

Fig. 2. IEEE 1641 signal defined in *OperationSetup* for an RS-232 bus

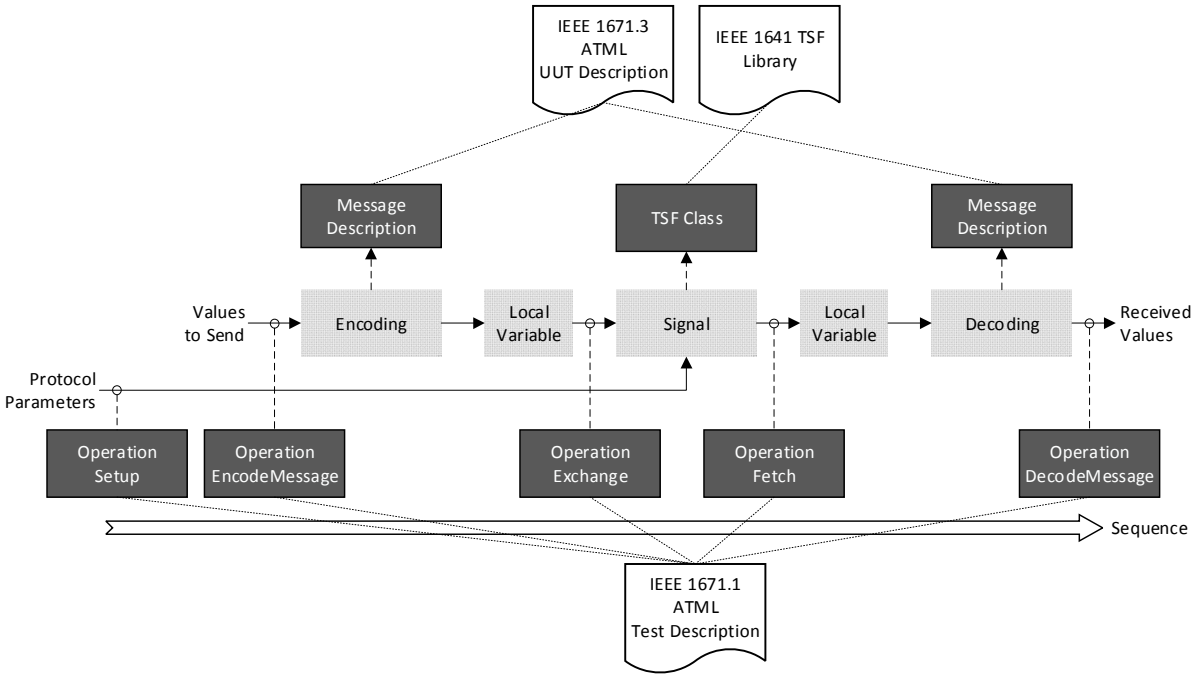

Fig. 4. Describing a bidirectional bus exchange with encoding and deconding using *Operations* and related elements

4. *OperationFetch* reads the received string from the signal attribute representing the "receive" data payload and stores it in a *LocalVariable*.
5. *OperationDecodeMessage* reads the data from the *LocalVariable*, decodes it according to a *MessageDescription*, and stores the floating-point value in a variable or transfers it to a test result.

For detailed information on various encoding formats refer to Annex C of IEEE 1671.3-2017.

D. Using Bus Test Operations with Attributes

As indicated earlier in the paper, the protocol parameters with constant or common values should be specified in the ATML UUT Description document, through *Attribute* elements assigned to *Bus*, *Node*, or *Message* elements. The values of *Attributes* can be retrieved through a specialized *Operation*, *OperationReadAttribute*.

RS-232: A test procedure contains multiple tests that communicate with the UUT through an RS-232 bus. The bus protocol is configured identically in all tests. To avoid repeating the same values in multiple *OperationSetup* operations, the values of protocol parameters are stored in *Bus/Attribute* elements and retrieved using *OperationReadAttribute*, as *illustrated* in Fig. 5.

VII. Conclusions

The paper shows how the operation and testing of digital serial buses can be described efficiently through the combined use of modeling elements from IEEE 1641 TSF Libraries, IEEE 1671.3 ATML UUT Description and IEEE 1671.1 ATML Test Description.

The models can be simple and compact for common bus applications, yet flexible enough to accommodate a variety of use cases, including fault insertion for testing the response of UUT bus interfaces in the presence of simulated transmission errors.

The design of IEEE 1641 and ATML schemas, along with consistent application of modeling guidelines proposed in this paper, allows organizations to develop reusable libraries of signal, bus, and test models.

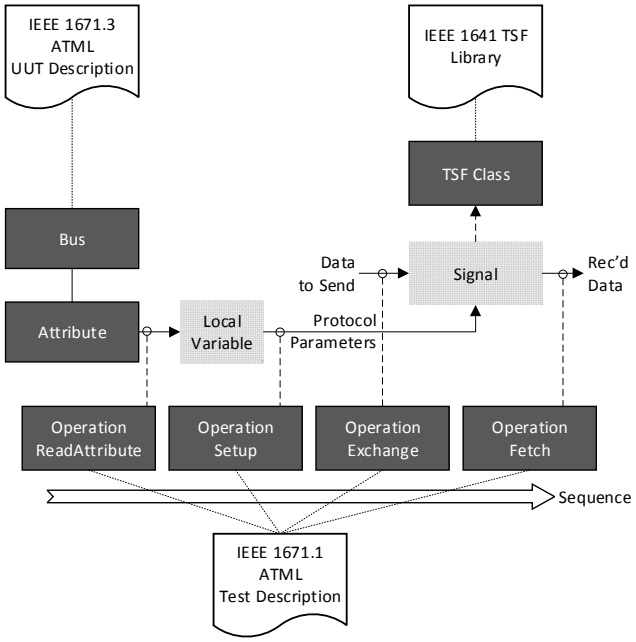

Fig. 5. Using *Attributes* to store comon protocol parameter values

ACKNOWLEDGMENT

While developing the techniques described in this paper the authors had productive discussions with Jean-Christophe Hertzog of MBDA.

REFERENCES

[1] *IEEE Std 1671.3-2017 - IEEE Standard for Automatic Test Markup Language (ATML) Unit Under Test (UUT) Description*. IEEE, 2017.

[2] *IEEE Std 1671.1-2017 - IEEE Standard for Automatic Test Markup Language (ATML) Test Descriptions*. IEEE, 2017.

[3] *IEEE Std 1641-2010 - IEEE Standard for Signal and Test Definition.* IEEE, 2010.

[4] L. Lindstrom, I. Neag, "Reducing Test Program Costs Through ATML-based Requirements Conversion and Code Generation", in *Proc. IEEE AUTOTESTCON*, 2013

[5] M. Cornish, M. Brown, A. Jain, T. Lopes, "An Open Source Software Framework for the Implementation of an Open Systems Architecture, Run-Time System", in *Proc. IEEE AUTOTESTCON*, 2012

[6] *IEEE Std 1641.1-2013 - IEEE Guide for the Use of IEEE Std 1641, IEEE Standard for Signal and Test Definition.* IEEE, 2013.

[7] *IEEE 1641.1a-2018 - IEEE Guide for the Use of IEEE Std 1641, IEEE Standard for Signal and Test Definition - Amendment 1: Addition of Guidelines for Producing Reusable Test Signal Frameworks for Use on Platforms Utilizing Automatic Test Markup Language*. IEEE, 2018.

[8] C Gorringe, "Bus testing in a modern era", in *Proc. IEEE AUTOTESTCON*, 2013.