Developing a Test Suite for Evaluating IEC 61499 Application Portability

Midhun Xavier*, Tatiana Laikh*, Sandeep Patil*, Valeriy Vyatkin* ‡
* Department of Computer Science, Computer and Space Engineering, Lulea Tekniska Universitet, Sweden
‡Department of Electrical Engineering and Automation, Aalto University, Espoo, Finland

Email: midhun.xavier@ltu.se, tatiana.laikh@ltu.se, sandeep.patil@ltu.se, vyatkin@ieee.org

2023 IEEE 32nd International Symposium on Industrial Electronics (ISIE) | 979-8-3503-9971-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISIE51358.2023.10228154

***Abstract*—This paper presents the creation of a series of function blocks with the specific aim of testing the portability of IEC 61499 applications across diverse development and runtime environments. These function blocks have been developed to cover a wide range of test scenarios, including basic data types, functions, boundary conditions, and adapter features. The function blocks can be conveniently exported or imported through the use of XML files, thus facilitating seamless testing. By testing the runtime environment of different IEC 61499 systems, these function blocks help to identify and highlight any possible issues that may arise related to portability.**

***Index Terms*—Industrial Control System, IEC 61499, Testing, Portability**

I. Testing IEC 61499 Applications: Ensuring Portability Across Different Environments

The IEC 61499 standard [1] has been developed to address the increasing demands for decentralized control and the exponential growth of control complexity in industrial automation systems, with the aim of establishing an open, component-oriented, and platform-independent development framework to improve the re-usability, reconfigurability, interoperability, portability [2], and distribution of control software for complex distributed systems. The IEC 61499 technology can be effectively used in the implementation of Intelligent Mechatronic Components and engineering processes, using multiple commercial tools and hardware platforms [3].

The paper [4] highlights the importance of addressing portability issues in existing engineering tools for the IEC 61499 standard in order to achieve better interoperability and efficiency in distributed control systems. Testing IEC 61499 applications for adequate functionality is crucial in guaranteeing their compatibility across various development and runtime environment [5]. To this end, a function blocks library has been designed for testing the basic data types, functions, and boundary conditions, among other key scenarios. A guideline document has been provided to help developers understand the desired results of testing and to modify the test with new values if necessary.

By enabling developers to test their IEC 61499 applications for portability using the developed standard testing function blocks, the likelihood of errors can be minimized and the efficiency and effectiveness of distributed industrial control systems can be improved.

II. Test Function Block Design and Development

A. Data Type testing FBs

1) BitStringDataType: The BitStringDataType function block (FB) is used to test the support of data types and their association for Bit data types, including BOOL, BYTE, WORD, DWORD, and LWORD. The testing procedure for the BitStringDataType FB involves two events, INIT and REQ.

When the INIT event is triggered, the output values of the BitStringDataType FB are set to predefined values. If the output values are not the same as predefined values then the test fails. This simple test ensures that FB can correctly handle each of the supported data types.

Alternatively, when input data is provided, the REQ event is triggered. The output values of the BitStringDataType FB should correspond to the input values. For example, Out1 should be the same as In1. The CNF event is also triggered by this test. This test ensures that the FB can correctly process the input data and produce the correct output values. The interface, ECC, INIT algorithm, and REQ algorithm for the BitStringDataType function block are depicted in figure 1 a, b, c, and d, respectively.

The BitStringDataType FB is an important testing tool for ensuring that data types and their associations are properly supported in Bit data types. By following the testing procedure outlined above, developers can ensure that their software systems can correctly handle different data types

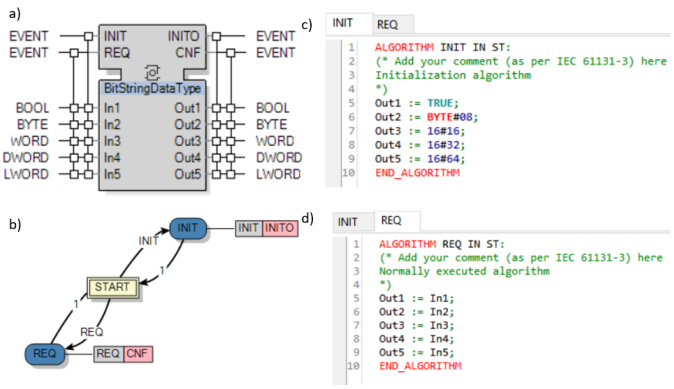

Fig. 1. a) BitStringDataType FB interface b) ECC c) INIT Algorithm d) REQ Algorithm.

and produce the expected output values.

2) IntegerDataType: The IntegerDataType function block is used to test the support and association of different integer data types, namely SINT, INT, DINT, LINT, USINT, UINT, UDINT, and ULINT, in an IEC 61499 application. Testing of the IntegerDataType function block can be done in manual mode by triggering the INIT event. When this event is triggered, the output values of Out1 to Out8 should be as follows: 10, 20, 30, 16#40, 16#50, 60, 70, and 80. This indicates that the function block correctly supports and associates the specified integer data types.

Another way to test the function block is to add values to the input data and then trigger the REQ event. In this case, the output values of Out1 to Out8 should have the same corresponding values as the input data. For example, the value of Out1 should be the same as the value of In1. This will also trigger the CNF event. These testing procedures help ensure that the IntegerDataType function block works correctly and supports the specified integer data types as expected. IntegerDataType FB interface, ECC, INIT algorithm and REQ algorithm is shown in figure 2 a, b, c and d respectively.

3) RealDataType: RealDataType also tests similar like the above test FBs. It is a function block designed for testing the data type support and association with real data types, REAL and LREAL. In the manual testing mode, triggering the INIT event would produce the following output and cause the INITO event to be fired: Out1 with a value of 10.100 and Out2 with a value of 20.2000. Alternatively, as in the case of the BitStringDataType and IntegerDataType test FBs, input data values can be added and the REQ event can be triggered. This would cause the output values to correspond to the input values, such as Out1 being the same as In1. This would also trigger the CNF event.

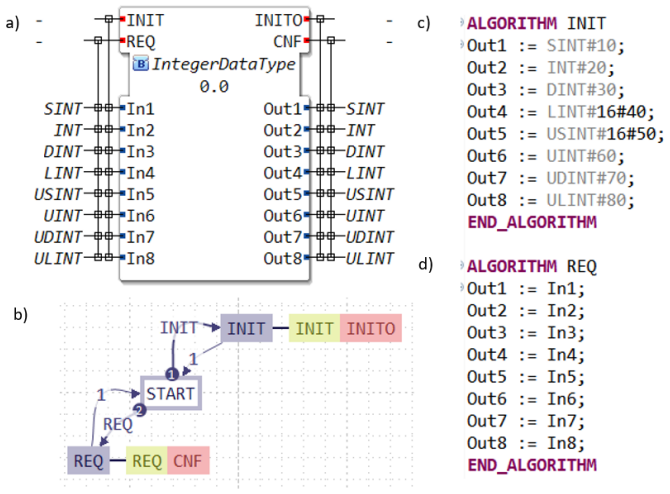

Fig. 2. a) IntegerDataType FB interface b) ECC c) INIT Algorithm d) REQ Algorithm.

4) StringDataType: The StringDataType function block (FB) is designed to test the support and association of the STRING data type. In manual testing mode, initiating the INIT event will produce the following output and trigger the INITO event:

Out1 := 'STRING'

Alternatively, adding values to the input data and triggering the REQ event will cause the output values to correspond to the input values. For example, Out1 should match the value of In1. This will also trigger the CNF event.

5) TimeDataType, DateDataType, TimeOfDayDataType and DateAndTimeOfDayDataType: The TimeDataType, DateDataType, TimeOfDayDataType, and DateAndTimeOfDayDataType function blocks (FBs) test the data type support and associations for their respective data types (TIME, LTIME, DATE, LDATE, TIME_OF_DAY, LTIME_OF_DAY, DATE_AND_TIME, and LDATE_AND_TIME).

In a manual testing mode, triggering the INIT event will produce output values, and the INITO event will be fired. For TimeDataType, the output values are Out1 := T#90s15ms and Out2 := LT#90s15ms542us15ns. For DateDataType, the output values are Out1 := D#1970-01-01 and Out2 := LD#2177-11-30. For TimeOfDayDataType, the output values are Out1 := TOD#00:00:00 and Out2 := LTOD#00:00:01. For DateAndTimeOfDayDataType, the output values are Out1 := DT#1970-01-01-00:00:00 and Out2 := LDT#1971-01-01-00:00:00.

Alternatively, input data can be added and the REQ event can be triggered. The output values should correspond to the input values, and the CNF event will be fired.

B. BoundCheckTest FB

BoundCheckTest FB 3 is a test function block that tests the boundary support for each data type such as BYTE, WORD, DWORD, LWORD, USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, LREAL, TIME, DATE, TOD, STRING. This function block can be used to check if the system under test can handle the maximum and minimum values of the specified data types correctly.

To test the BoundCheckTest function block, input data needs to be added with necessary values and then the bound check event for the specified data type should be triggered. The output values of the event should be the input value plus one. For example, if the input value is In1, the output value should be Out1 = In1 + 1. This will also trigger the CNF event, indicating that the test has been completed successfully.

The purpose of the BoundCheckTest is to ensure that the system under test can handle the maximum and minimum values of the specified data types without any errors or unexpected behavior. This is important because it ensures the reliability and robustness of the system, especially in situations where the system is required to handle extreme values.

C. StanadardFunctionTest FB

The StandardFunctionTest FB is designed to evaluate the support of IEC 61499 standard functions and their correspond-

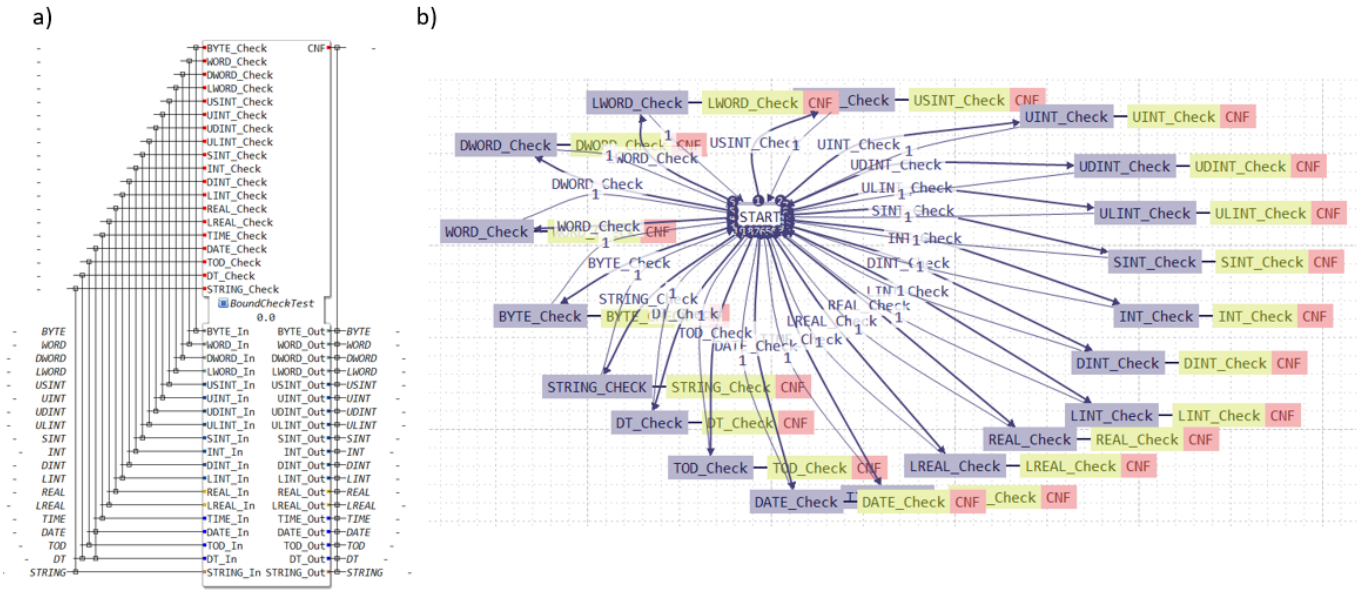

Fig. 3. BoundCheckTest FB a) Interface b) ECC

ing association with input values. This FB is capable of testing over 75 standard functions by providing specific events for testing each function, which trigger the corresponding testing algorithm for that particular function.

To test the StandardFunctionTest function block, suitable input values are added to the input data, and the function event is triggered. The output values should contain the result of the corresponding function, which is then verified by the CNF event.

For instance, if In1 is the input value, the output value Out1 should be equal to the result of the function evaluated using In1. Therefore, the CNF event confirms that the expected output value is produced. Let's consider the following example to test the MUX function feature,

To test the Multiplexer (MUX) function, the MUX_Func event should be triggered, and the corresponding algorithm will execute.

```
ALGORITHM MUX_Func IN ST:
(* Add your comment (as per IEC 61131-3) here
*)
MUX_Out:=MUX(int1,time1,time2,time3);
END_ALGORITHM
```

The MUX function is used to select one of several input signals based on an index value. The input signals can be of different types, such as time, integers, etc. The MUX function returns the selected input signal based on the index value. The MUX function can be tested by triggering the MUX_Func event and verifying that the output value matches the expected output value.

If we provide input values int1=2, time1=TIME#1m30s, time2=T#15s, and time3=T#2m30s, then the expected output

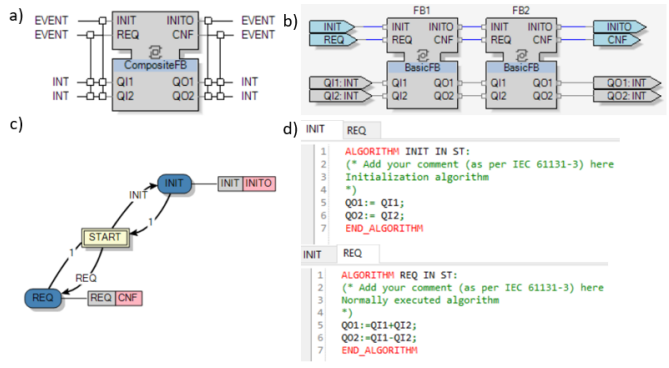

Fig. 4. a) Composite Function Block a) FB interface b) Basic FB connection Diagram c) ECC d) INIT Algorithm and REQ Algorithm.

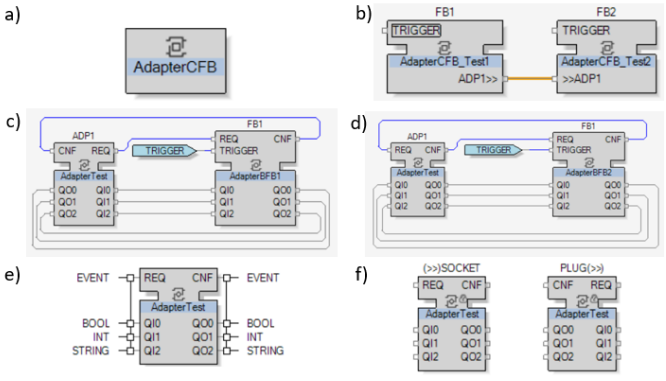

Fig. 5. shows the Function Block Interface of a) Adapter CFB b) AdapterCFB consists of two sub-CFBs i.e. AdapterCFB_Test1 and AdapterCFB_Test2 c) AdapterCFB_Test1 consists of AdapterTest FB and an AdapterBF1 (basic FB) d) AdapterCFB_Test2 consists of AdapterTest FB and an AdapterBF1 (basic FB), e & f) Represents the AdapterTest FB and its SOCKET and PLUG

is MUX_Out:=MUX(int1,time1,time2,time3) = 2m30s.

D. CompositeBlockTest

The CompositeBlock FB 4 is used to test the functionality of composite function block support. The block has two events, INIT and REQ, which can be triggered in manual testing mode or by adding integer values to the input data and triggering the REQ event.

When the INIT event is triggered, the output values should be the same as the input values, and the INITO event is fired. On the other hand, when the REQ event is triggered, the output should be calculated as QO1 = QI1 + QI2 and QO2 = QI1 - QI2. After the calculation is complete, the CNF event is fired.

In summary, the CompositeBlock function block tests the functionality of composite function blocks by calculating the output values based on the input values and triggering the CNF event when the calculation is complete.

E. AdapterTest CFB

AdapterCFB 5 is a function block that tests the functionality of adapter support. The AdapterCFB is composed of two basic function blocks, AdapterBFB1 and AdapterBFB2. AdapterBFB1 has three output parameters, QO0, QO1, and QO2, which are initialized to FALSE, 1, and "AdapterBFB1", respectively. AdapterBFB2 also has three output parameters, QO0, QO1, and QO2, which are initialized to TRUE, 2, and "AdapterBFB2", respectively.

The input values from AdapterBFB1 are transferred to AdapterBFB2 and vice versa. This is made possible through the use of an adapter, which connects the two function blocks. When the AdapterCFB is triggered, the input values are set by the tester, and the output values are calculated by the adapter based on the input values.

In other words, AdapterCFB provides a way to connect two function blocks that have different inputs and outputs by mapping the inputs and outputs of one block to the inputs and outputs of the other block, using an adapter.

III. XML-IMPORT/EXPORT COMPATIBILITY TEST BETWEEN 4DIAC V2.0.1 AND EAE V21.2

As a part of the 1-Swarm project, a hackathon was conducted to analyze the compatibility of IEC 61499 between 4DIAC and Schinder Electric EcoStruxure (EAE). The objectives of the hackathon were to identify portability issues between 4DIAC and EAE-IDE, as well as between Forte and EcoRT-runtime, create a roadmap for addressing the identified issues, and discuss potential solutions for creating a single portable test application that can automatically test new versions of IDE and runtime and certify some key attributes. Some of the issues identified during the hackathon are given below:

1) Issue: The EAE import dialog does not allow for the selection of a $< BitStringDataType.Basic.ZIP >$ named file with a subdirectory IEC61499 and a metafile named $< BitStringDataType.Basic.export >$. EAE requires a specific extended import format which is not described in the IEC61499 standard.
   Recommendation: Develop a company conformance profile or allow for the import of pure .fbt files to address the issue.
2) Issue: After importing a 4diac .xml file in the 4DIAC .BASIC.ZIP format, the algorithms are not imported. EAE does not accept the CDATA format for the textual contents of algorithms.
   Recommendation: Modify EAE to accept CDATA format as it offers user format overall lines, instead of compressing the algorithm in one .xml line.
3) Issue: The compiler/parser does not generate an error when a function with two parameters is called with only one argument.
   Recommendation: Modify the compiler/parser to generate an error when an argument is missing.
4) Issue: EAE does not support data types CHAR, WCHAR, and WSTRING. Only the STRING data type is supported.
   Recommendation: Modify EAE to support character string literals that directly represent a character or character string value of data type CHAR, WCHAR, STRING, or WSTRING as documented in the IEC61131-3 standard.

An automated testing approach for assessing the compatibility of software tools and runtime platforms with the IEC 61499 standard could be a promising direction for future research and development. This approach could leverage service sequence testing as an additional testing methodology to ensure comprehensive compliance with the IEC 61499 specification across all integrated development environments (IDEs) and runtime environments.

ACKNOWLEDGMENT

This work was sponsored, in part, by the H2020 project 1-SWARM co-funded by the European Commission (grant agreement: 871743). Thank you Artur Fritz and Alois Zoitl for their valuable contributions and participation in the hackathon for the development of the IEC 61499 portability test suite.

REFERENCES

[1] "IEC 61499-1: Function Blocks Part 1: Architecture," 2012.
[2] C. Gerber and H.-M. Hanisch, "Does portability of iec 61499 mean that once programmed control software runs everywhere?" *IFAC Proceedings Volumes*, vol. 43, no. 4, pp. 24–29, 2010.
[3] S. Patil, J. Yan, V. Vyatkin, and C. Pang, "On composition of mechatronic components enabled by interoperability and portability provisions of iec 61499: A case study," in *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*. IEEE, 2013, pp. 1–4.
[4] C. Pang, S. Patil, C.-W. Yang, V. Vyatkin, and A. Shalyto, "A portability study of iec 61499: Semantics and tools," in *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*. IEEE, 2014, pp. 440–445.
[5] A. Hopsu, U. D. Atmojo, and V. Vyatkin, "On portability of iec 61499 compliant structures and systems," in *2019 IEEE 28th International Symposium on Industrial Electronics (ISIE)*. IEEE, 2019, pp. 1306–1311.