Hopsu, Alexander; Atmojo, Udayanto Dwi; Vyatkin, Valeriy

On Portability of IEC 61499 Compliant Structures and Systems

Published in:
Proceedings of the 2019 IEEE 28th International Symposium on Industrial Electronics, ISIE 2019

DOI:
10.1109/ISIE.2019.8781290

Published: 01/06/2019

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Hopsu, A., Atmojo, U. D., & Vyatkin, V. (2019). On Portability of IEC 61499 Compliant Structures and Systems. In *Proceedings of the 2019 IEEE 28th International Symposium on Industrial Electronics, ISIE 2019* (pp. 1306-1311). Article 8781290 (Proceedings of the IEEE International Symposium on Industrial Electronics). IEEE. https://doi.org/10.1109/ISIE.2019.8781290

On Portability of IEC 61499 Compliant Structures and Systems

Alexander Hopsu[1], Udayanto Dwi Atmojo[1], Valeriy Vyatkin[1,2]
[1] Department of Electrical Engineering and Automation, Aalto University, Espoo, Finland
[2] Dept. of Computer Science, Electrical and Space Engineering, Luleå University of Technology, Luleå, Sweden

alexander.hopsu(at)aalto.fi, udayanto.atmojo(at)ieee.org, vyatkin(at)ieee.org

***Abstract*—This paper investigates the portability features of three different IEC 61499 standard compliant tools. The study focuses on migrating the basic and composite function block types and system architecture with application networks and device configurations from one tool to another. A converter program is subsequently created using Python programming language to automate the required modification process, thus enabling the files to migrate between the compliant tools. The study takes into consideration NxtStudio, FBDK and 4DIAC software tools. In every tool, similar function blocks and system structures are created. The portability of these created elements is examined between the tools, resulting in a table that numerically evaluates the portability from one tool to another.**

Keywords— IEC 61499 standard, portability, compliant tool, converter program, NxtStudio, FBDK, 4DIAC

I. INTRODUCTION

Automation is the heart of automated assembly lines which reduces costs while keeping quality of the products within acceptable level. Modern manufacturing utilize Programmable Logic Controllers (PLCs), which are the key elements for enabling the operation of automated factories. Production is at the most effective level when the machines are constantly running without any disturbances. However, failures and reconfigurations of the assembly lines are inevitable at some point, due to the deteriorating condition of the controllers and machineries over the time and the changing demand from customers (e.g., product customization). These lead to down-time which costs money and resources to the company.

The IEC 61131-3 standard is one of the most used programming technologies for PLCs [1]. IEC 61131-3 [2] is aimed for centralized control architecture. Thus, systems based on IEC 61131-3 are prone to single-point of failure. In addition, the IEC 61131-3 standard does not offer an easy and comprehensive method for reconfiguration of the system in different situations [3]. The IEC 61499 standard [4] resolves these issues with the concept of application-centric, distributed control design. With decentralized control, single point of failure is no longer an issue, where in case of failures, production may still progress to some extent without a complete halt. In IEC 61499, software control logic can be easily redeployed in different hardware controllers in case one of them fails. Meanwhile, the IEC 61499 standard is based on event driven execution model, where the parts of the software are executed only when they are invoked. The standard differs from the cyclic execution model of IEC 61131-3. The event-driven approach of IEC 61499 allows the execution of only certain control logic which is triggered by incoming events, thus allowing computational resources to be effectively utilized, i.e., only for control logic "activated" by incoming events. The base element of the standard is a function block (FB), which has event and data inputs and outputs. Each FB is activated by triggering one of its event inputs, followed usually by one or more event outputs for triggering the next FBs. An IEC 61499 systems are usually composed of a network of FBs, which can be mapped to one or more devices.

One of the main aims of the IEC 61499 standard is to make the control software easily portable and reconfigurable between devices of different vendors. To achieve this, vendors are expected to follow the compliance profile rule strictly. However, this doesn't seem to be the case, as this work will demonstrate. This paper investigates the portability of IEC 61499 systems among different IEC 61499-compliant software tools. The paper concentrates particularly on three different IEC 61499 compliant tools and tries to address issues relating to portability between them. Based on the investigation, a Python-based converter program is created to automate the required modifications on the source files to enable and increase the portability between the compliant tools.

II. RELATED WORKS

A. IEC 61499 standard

The IEC 61499 standard has been developed to solve the limitations of the IEC 61131-3 standard. It enables the design of distributed, control in industrial automation. The standard uses function blocks (FBs), which are event-driven [5]. In IEC 61499, each FB consists of event and data interfaces for both inputs and outputs (see Figure 1, upper left). A basic FB has Execution Control Chart (ECC), which consists of a set of finite state machines (See Fig. 1, lower). The initial state of ECC is the "START" state, and at any given time, the current state can change with respect to the conditions defined in the transition branches between the states. Each state (except START) can perform certain operations defined as algorithms, which can be implemented in various different programming languages (See Fig. 1, upper right), e.g., Structured Text. During execution, a FB can be monitored to show the value of its input and output. In Fig.1 (top left), a FB named "Calculate" is monitored and the snapshot of its current values is shown. In this example, the event CALC is invoked once with data values OPER = '+', VALUE1 = 3 and VALUE2 = 5, which triggers the transition of the ECC from START into SUM state. In SUM state, the associated algorithm SUM is executed, which outputs data through the RESULT output data interface and event output

CNF associated to the data, then the FB returns to START state. If any output event is generated, the event output can be transferred to the next FB connected to the output event.

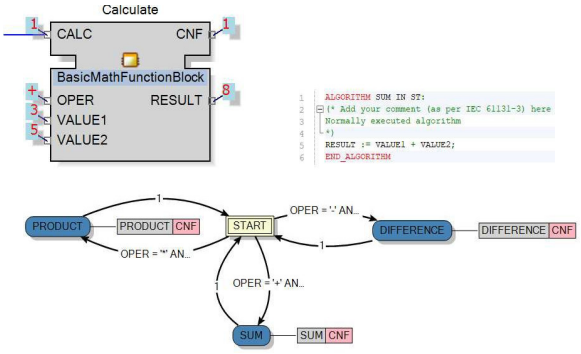

Fig. 1. IEC 61499 basic function block, internal ECC state machine and ST-language algorithm SUM implemented in NxtStudio.

In addition to the basic FB, there are also composite and service interface FBs. Composite function block encapsulates a network of FBs and also has inputs and outputs. The use of composite FBs enables hierarchical software design, which simplifies the structure of FB applications. On the other hand, service interface FBs are considered as "black boxes", whose internal structure is not very strictly defined. They can implement, e.g., different communication interfaces/protocols.

The design process of an IEC 61499 system consists of two parts: application and system configuration. The application contains the network of the function blocks and their connections, which capture the overall software logic of the program. The system configuration defines how FBs in the application are mapped to control device(s). This allows for flexibility in managing computational resource allocation. For example, FBs which have computationally demanding algorithms should naturally be mapped to the device which is computationally capable compared to other devices. To optimize this mapping of the FBs on devices, the project TORERO [9] has created an automatic mechanism to define the system configuration. It takes into account, for example, the computational power required for steps of function blocks, application real-time constraints, available device memories and communication protocols of the application. Based on these, the TORERO software automatically creates the required communication function blocks based on the system configuration [6].

B. Software portability

Software portability enables (at least part of) the software to be reused in other software tools or operation environments ("Write Once Run Anywhere/Everywhere"). By porting the same software from one environment to another, the effort for modification or adaptation of the software for the new environment is reduced (even annulled), therefore reducing cost [7] [8] [9]. The IEC 61499 standard itself aims to make control software portable. In the first edition of the standard, this was not quite achieved because the execution semantics of the function blocks were not defined accurately enough (ambiguous) [10] . For example, the lifetime of an input event was not defined, which caused difference in execution behaviours between different software tools. Due to such difference, one IEC 61499 system designed by one IEC 61499 compliant tool may not be ported to other tools without certain modifications.

Software tool	FBDK	4DIAC	NxtStudio	ISaGRAF
FBDK	Full	Full	Partial	Not applicable
4DIAC	Full	Full	Partial	Not applicable
NxtStudio	Partial	Partial	Full	Not applicable
ISaGRAF	Not applicable	Not applicable	Not applicable	Full

Table 1. Portability of the library elements between different IEC 61499 compliant tools [11].

In the second edition of the IEC 61499 standard, the execution semantics have been better defined. However, there are still existing portability issues between different IEC 61499 tools, e.g., Function Development Kit (FBDK) [12], ISaGRAF Workbench [13], 4DIAC [14], and NxtStudio [15], just to name a few. According to the standard, the storing format of the library elements in IEC 61499 is XML. FBDK and 4DIAC are following this guideline very strictly, thus FB library developed using the two are portable with each other. NxtStudio has some own additional library features, such as Composite Automation Type (CAT) function blocks, which include the control, visualization and plant model parts in one function block. These special FBs and other NxtStudio have specific attributes in the XML structure which are not defined in other tools. In contrast, ISaGRAF Workbench has a different format for storing FB library elements, which makes them totally non-portable to other IEC 61499 compliant tools.

Currently, only few studies regarding the portability of IEC 61499 tools have been done. Some examples are [11], which presents a summary shown in Table 1 and [16] demonstrates some extent of provisions to port IEC 61499 FBs on different tool environment in a case study. However they focused mostly on the FB aspect. This paper investigates the portability of IEC 61499 tools beyond the scope of the FB itself, which includes the IEC 61499 system configuration. The paper also presents some metric of portability between the IEC 61499 tools and proposes a new software solution to ease the programmers' effort to port IEC 61499 system between different IEC 61499-compliant software environments.

III. PORTABILITY OF IEC 61499 SYSTEMS BETWEEN TOOLS

In this work, a traffic light example was developed using different IEC 61499 tools, i.e., FBDK, NxtStudio, and 4DIAC, to investigate the exact causes leading to portability issues of IEC 61499 tools (see Fig. 2). It's important to note that in this paper, we consider only NxtStudio version 2.1, FBDK version 2.6, and 4DIAC version 1.8.4 in Windows 10 operating system. The example consists of four traffic light units, whose control software are distributed on four different devices, in this case, simulated/"soft" controllers. A "master control" function runs on the fifth device, which determines the current mode of all traffic lights. Each traffic light can be in either of the following modes: set to run on 2 or 4 phase cycle mode, set to red, turned completely off, or set to blinking mode.

In NxtStudio, when FBs of the application are mapped to multiple devices, NxtStudio automatically establishes the needed communication interfaces between these distributed FBs. This is not the case with FBDK and 4DIAC, where the tools do not automatically create the communication interfaces, and thus the programmers need to add communication FBs themselves. In FBDK, communication interface is automatically created by the tool when the PUBL and SUBL type function blocks have the same name in different resources. However, by merely connecting the event and data interfaces of the mapped elements (in different devices) in the application level does not make the tool to automatically create the communication interface, as in the case of NxtStudio and 4DIAC. In 4DIAC, the example has five more devices in order to deploy graphical (e.g., HMI) application for each traffic light unit and master controller. The traffic light control logic runs on five FORTE devices, while the graphical applications utilize FBRT (Function Block Runtime) [12] and execute on the other five devices. Thus, the 4DIAC example is distributed over ten devices.

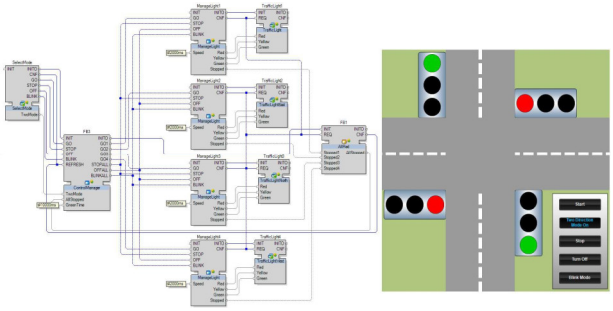

Fig. 2. Traffic light application implemented in NxtStudio.

Because of the relatively high occurrence of event transmissions instead of data in this case study, each event requires its own publisher and subscriber FB in FBDK and 4DIAC, which need to be created manually, whereas NxtStudio handles the communication between different devices automatically. Therefore, the use of the communication blocks especially in 4DIAC with 10 devices requires major effort, as the IEC 61499 standard's library does not offer simple solution to handle large number of transmittable events with only one or few communication blocks. Additionally, the inability to modify and compile the "FB network" inside composite function block in FBDK causes unnecessary extra number of FBs to be shown in the application / device resource.

To investigate the portability between tools, this work makes use of the following approach. The traffic light example is implemented using each software tool, then we attempt to open and run the implementation (which includes the FB application and system configuration) without any modifications using different tools. When problems are encountered, we examine to find the underlying causes, and then come up with the way to mitigate. The portability study in this work is limited only "simulated/soft" devices which runs on PC. Portability between tools in different setting (e.g., deployment on "real" hardware control devices) is out of the scope of this work. The following describes our findings and solutions to mitigate the problems we faced during the portability investigation.

A. Portability from NxtStudio to FBDK

Importing the [SystemName].sys file, which is the IEC 61499 system configuration, can be done by drag-and-dropping the file from the "[NxtStudio's project folder]/IEC61499/System" folder into the editor area of FBDK. Before FBDK can open the file, some elements in the file which are not supported by NxtStudio (such as AvoidsNodes and Points) need to be removed. We found that FBDK can automatically remove NxtStudio specific XML attributes the system configuration file (e.g., "NameSpace"). In order for correct mapping of the FBs to the devices, the name (identifier) of the FB has to be included in the mapping element after the resource name in the attribute "To".

In NxtStudio, the device type of simulated device (SoftPLC) is NXT_RMTDEV, which has a resource named EMB_RES_ENH that can run FBs for HMI. Meanwhile in FBDK, simulated device is named FRAME_DEVICE, and it comes with resource called PANEL_RESOURCE that can run FBs for HMI. Implementation of HMI in NxtStudio and FBDK is different, thus all software elements for HMI in both tools are not compatible/portable with each other. Meanwhile, the system configuration file generated by NxtStudio can be opened in FBDK by removing unsupported elements in the file. NxtStudio automatically inserts communication FBs to enable the communication between FBs mapped to different devices. This is added in the XML structure of the system configuration file, however this addition is not recognized by FBDK. Thus, the XML elements associated to these communication FBs need to be removed and replaced by other FBs known by FBDK (e.g., the "PUBLISH" and "SUBSCRIBE" FBs).

B. Portability from FBDK to NxtStudio

One way to import the system file from FBDK to NxtStudio is to modify or change the XML elements in the system file. For example, the name of the device (FRAME_DEVICE) and resource in FBDK have to be changed into the name recognized/used in NxtStudio, i.e., NXT_RMTDEV and EMB_RES_ENH. Also, some NxtStudio specific XML attributes (e.g., "NameSpace") need to be included in the system file with their respective values. For example, in NxtStudio, the value of the "Namespace" attribute for FBs created by the user is "Main", whereas the value of simulated device (SoftPLC) NXT_RMTDEV is "nxtControl.Standard". The value for the resources of the default devices is "Runtime.Management", and the communication FBs is "IEC61499.Communication".

If the application has SUBL and PUBL communication FBs (in FBDK), these FBs are not recognized in NxtStudio, and thus need to be removed, or later changed for example to SUBSCRIBE and PUBLISH FBs when the application is opened in NxtStudio. Furthermore, the port number configuration used for deployment in either NxtStudio's Devices tab or the configuration file named [SystemName].cfg and [SystemName].Device.properties have to be defined properly so the IEC 61499 system can run.

C. Portability from NxtStudio to 4DIAC

Importing NxtStudio into 4DIAC editor is done via the 4DIAC's own import feature File -> Import -> 4DIAC ->

System Import. Through the import feature, 4DIAC requests for the system file and the FB files. During the import process, the tool will notify whether the import is successful or certain modifications need to be done.

Similar to porting from NxtStudio to FBDK, when porting from NxtStudio to 4DIAC, the communication FBs originally inserted by NxtStudio need to be removed and replaced by some other communication FBs recognized by 4DIAC. Also, any NxtStudio HMI FBs need to be completely removed or replaced by the ones supported by 4DIAC. However, 4DIAC can automatically remove NxtStudio specific unsupported XML elements and their attributes from the system file, such as AvoidsNodes, Attribute, and NameSpace. The default device type NXT_RMTDEV and resource type EMB_RES_ENH of NxtStudio need to be changed into the 4DIAC's FORTE_PC and EMB_RES to run the non-graphical (non-HMI) functionality. NxtStudio graphical FBs (e.g., CAT FBs) are not compatible with 4DIAC. 4DIAC can use, e.g., FBDK's FBRT_WINDOW device containing PANEL_RESOURCE resource to run any graphical functions. We found that during the import process, 4DIAC will add an extra empty default resource type alongside the existing resource type(s) into the devices.

D. *Portability from 4DIAC to NxtStudio*

Importing the system configuration file from 4DIAC to NxtStudio requires similar steps as in the case of importing from FBDK to NxtStudio. The system configuration file can be opened by NxtStudio, however it is strictly read-only. To actually run the program, a system file needs to be created using NxtStudio, and then has its content modified/replaced with the one created in 4DIAC.Then, the device and resource types (FORTE_PC or FBRT_WINDOW and EMB_RES or PANEL_RESOURCE) have to be changed to the ones recognized by NxtStudio (NXT_RMTDEV and EMB_RES_ENH). Also, parameters which are not known in NXT_RMTDEV's interface, e.g., "Color" and "Profile" which are originally from and specific to 4DIAC, need to be removed.

Furthermore, the system configuration file has to contain the "Namespace" attribute with the proper value for every FB, devices and their respective resources. The value "Main" is for user-created function blocks, "nxtControl.Standard" for "SoftPLC"/simulated devices, "Runtime.Management" for their resources, and "IEC61499.Communication" for communication FBs. Finally, the device configuration (such as port number) has to be set properly before the IEC 61499 system can run.

E. *Portability from FBDK to 4DIAC*

Porting from FBDK to 4DIAC is done using 4DIAC import. The system configuration file created by FBDK does not need to be modified, except for the device and resource types which need to be "FORTE_PC" with "EMB_RES" resource. In case when FBDK's graphical elements are used, the device and resource type should be changed into "FBRT_WINDOW" with "PANEL_RESOURCE" if using simulated device. 4DIAC also adds extra empty default resource to the devices and duplicates the mapped FBs to the resource when the system configuration file is imported from FBDK. This is shown in Fig.3. After the import, it is possible that connections between mapped FBs in the resource view becomes incorrect, at least from the graphical perspective. However, whether this will affect the behaviour of the system will require further investigation.

We found that the PUBL and SUBL FBs in FBDK do not work exactly the same way as 4DIAC. In FBDK, the SUBL block will only receive the event and data from the PUBL block in different devices if the names of the FB instances are identical. This is important to note for programmers when they decide to use PUBL and SUBL FBs and will have the system run on both FBDK and 4DIAC environments. Also, since 4DIAC only has the default library set from FBDK, 4DIAC will not be able to compile composite FBs developed in FBDK.

F. *Portability from 4DIAC to FBDK*

IEC 61499 systems developed in 4DIAC can be opened in FBDK by drag-and-dropping the file into the editor area of FBDK. Before FBDK accepts the file, some unsupported XML elements, e.g., "Color" and "Profile" attributes need to be removed. After that, FBDK accepts the file without any error messages. However, before execution, in this case, on simulated devices, the device and resource types in the system configuration file need to be changed into the ones used in FBDK, FRAME_DEVICE with PANEL_RESOURCE.

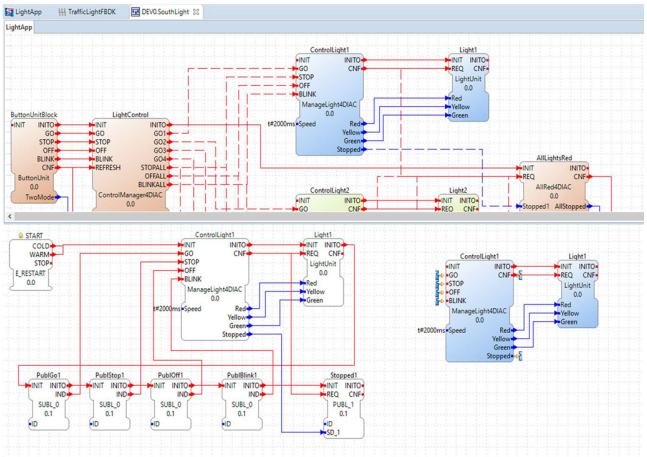

Fig. 3. System importing process in 4DIAC.

IV. RESULTS

Section III describes our findings when we attempted to port the traffic light example between different IEC 61499 tools. The porting attempts between tools was generally successful with either minor or greater modifications into the source code. This section briefly points several important points we found during our investigation and proposes a new tool to ease the effort in porting IEC 61499 system between different IEC 61499 tools.

A. *Portability assessment between the IEC 61499 compliant tools*

All three IEC 61499 compliant tools use the XML format, as defined in the standard, for saving the information regarding FBs and the system configuration. However, despite that all tools use XML, not all follow the standard strictly which causes portability issues between tools. Among one of the biggest portability issues between the tools is the porting of graphical

(e.g., HMI) functions. In NxtStudio graphical functions are programmed in C# encapsulated as CAT function blocks, whereas in FBDK they are implemented in Java. 4DIAC can use the graphical library from FBDK, but since some of the graphical elements in FBDK can be encapsulated inside of composite function blocks, this causes restrictions when porting them into 4DIAC, as the composite FBs designed in FBDK cannot be compiled neither into FORTE nor FBRT. This issue is expected, since IEC 61499 standard doesn't extend into graphical functions. Graphical functions can be regarded as SIFBs for the visualization device and SIFBs are tool/platform dependent.

Another issue, although doesn't necessarily affect the execution of IEC 61499 systems between the tools, is the definition of the horizontal and vertical locations of the function blocks in the tool editor area which differ between these tools. This is apparently caused by different coordinate systems of the platforms used for the IDEs implementation. In large applications, this might cause very messy visual rendering, as a group of FBs might be piled up on top of each other. The difference in how coordinates are defined between tools can cause FBs to go outside of the visible area of the tool editor, which might be challenging to notice. When this happens, some programmers would agree that having the feature of automatic tidying and positioning of FB within the editing area in the tool will help.

Different from FBDK and 4DIAC, NxtStudio includes a large number of XML elements specific only to NxtStudio. When an IEC 61499 system designed in NxtStudio needs to be ported to other tools, e.g., FBDK, many XML elements need to be removed before FBDK can accept. Meanwhile, 4DIAC is able to remove unsupported elements and save only those which are recognized and needed. Importing from FBDK or 4DIAC into NxtStudio is a bit tricky however, as NxtStudio does not provide a way for importing the system configuration file except only in a compressed (zip) format. When importing to NxtStudio, some XML attributes (e.g., "Namespace") are required for every function block in the application in order to be recognized by the tool. Additionally, since NxtStudio automatically introduce communication FBs between mapped FBs in different devices, in case of importing from FBDK, existing communication FBs required in FBDK needs to be removed. Finally, the suitable supported device and resource type needs to be defined accordingly on the destination/target tool.

Our findings on the portability investigation is summarized in Table 2. The table presents the portability rating between the tools based on the portability tests in chapter 3. According to our knowledge, we are not aware of any standardized methods to measure the portability features of different IEC 61499 compliant tools. The values are qualitative which are determined based on:

1) Required number of elements and their types to replace/modify inside the file (maximum 35%, the higher the percentage is, the less number to modify in the files),

2) Difficulty in replacing the element by other element or structure (such as graphical/HMI functionalities). The value is maximum 45 %, the higher the percentage is, the easier it is to replace the elements), and

3) The easiness in importing elements into the tool (max 20 %). The higher the value is, the less effort in addition to the normal importing process is required by the user to get the component or system to work in other software tool).

We found that when IEC 61499 systems involve the use of graphical functions (e.g., visual simulation of the "physical" system, HMI), they are more difficult to port to different tools.

Portability of IEC 61499 compliant tools in scale of 0 – 100 %		To		
		NxtStudio	FBDK	4DIAC
From	NxtStudio	100 % (35, 45, 20)	45 % (20, 10, 15)	50 % (25, 10, 15)
	FBDK	45 % (25, 10, 10)	100 % (35, 45, 20)	90 % (30, 40, 20)
	4DIAC	50 % (30, 10, 10)	95 % (30, 45, 20)	100 % (35, 45, 20)

Table 2. Degree of portability of IEC 61499 compliant tools based on the investigation in chapter 3. The numbers inside the brackets refer to the evaluation aspects (1, 2, 3) described above the table.

B. Converter program

Based on the findings, a converter program was developed in Python (library version 3.3.5) to automate the required modification effort (which otherwise needs to be done manually by programmers) to simplify porting of IEC 61499 system between the three tools. The converter can perform the modifications based on those identified from our test case.

```
Welcome to the IEC 61499 compliant tool converter.
This program converts the source files from one compliant tool to another.
The supported elements to be converted are:
[Basic function block, Composite function block, System file].

The supported compliant tools are:
1. NXT, version 2.1
2. FBDK, version 2.6
3. 4DIAC, version 1.8.4

Please select the source tool from which you want to convert by writing number (from 1 to 3):
1
Please select the target tool for which you want to convert by writing number (from 1 to 3):
2

Please write the path from where the XML source file will be read (using back '\' or forward '/' slashes).
If the path is a directory, all function block files (.fbt) in that particular folder
will be automatically converted from source tool to target tool written in new separate files:
C:/Users/Hopsale/Desktop/ControlLightNXT.fbt

Conversion was succesfully done.
Please select the operation for the converted XML file from NXT to FBDK:
1. Print XML tree on terminal.
2. Write XML tree on file.
2

Do you want to create the new file in some desired existing location? y: Yes, [other]: No.
```

Fig. 4. The user interface of the converter program running in Eclipse IDE console.

When executed, the converter program requests the user to select one (or more) actions from the available options by typing the selected option. Then, the user is asked to define which files to be processed by the converter, where they will be stored (path), and what the user wants to replace (for example, HMI FBs). The program consists total of 9 different Python modules, listed on Table 3. It uses Python default library set without any third-party libraries needed. The converter is started by running the MainProgram.py module, which has a "bootstrap" function and is responsible for executing other functions in the converter tool accordingly based on the inputs from the user. The program is able to process individual function block(s) or the system configuration file. The converter has built-in error handling functions to deal with, e.g., invalid structures of the XML file or invalid user input. The error-handling functions present information about the error if such is present to inform the user where the error comes from, and then the program will terminate.

Module	Function
MainProgram	Module acting as a "bootstrap" and execute other functions accordingly based on the inputs from the users
XML_Element	Class representing one XML element of the file.
XML_File	Class representing the entire XML tree of the file, containing list of XML_Element objects.
XML_Parser	Module reading and parsing the actual source files for constructing them as a XML_File object.
XML_Modifier	Module that execute functions provided by NXT_Functions, FBDK_Functions and FOURDIAC_Functions modules
Parser_functions	Module that provides supporting functions for the XML_Parser module.
NXT_Functions	Module containing all the necessary functions to convert files from NxtStudio to FBDK or 4DIAC. Contains also a zip packing manager for importing files into NxtStudio. (Python's zip compression method doesn't work with NxtStudio)
FBDK_Functions	Module containing all the necessary functions to convert files from FBDK to NxtStudio or 4DIAC.
FOURDIAC_Functions	Module containing all the necessary functions to convert files from 4DIAC to NxtStudio or FBDK.

Table 3. Modules of the converter program with functionality descriptions.

Also, we introduce similar compression feature like in NxtStudio, in the hope that once the tool generates compatible FB and system configuration files for NxtStudio, it can put them in a single compressed format for ease importing in NxtStudio. However, we found that none of the compression types offered by Python library, i.e., ZIP_STORED, ZIP_DEFLATED, ZIP_BZIP2 or ZIP_LZMA, works with NxtStudio. When converting the files to be imported into NxtStudio, the converter program asks the user, whether the files will be compressed by the converter program. In our experience, the compressed (zip) files created using Windows 10 zip compression tool may face certain issues if they have certain names when imported to NxtStudio. However, at this stage we haven't found any conclusive reasons why this happens.

V. Conclusions and Future Work

The main purpose of this work was to investigate the portability issues between different IEC 61499 compliant tools. The topic was first opened by introducing the features of the IEC 61499 standard, followed by justifications why portability is important. The paper considered three most utilized IEC 61499 tools NxtStudio, FBDK and 4DIAC to investigate. Features of the software tools related to portability were presented. We utilized a traffic light example to investigate the portability of IEC 61499 systems between the tools. Then, the portability of function block between the tools was examined. Based on this, we identified several modifications which are imperative to allow opening & executing FB in different tools. While in general the three tools comply with IEC 61499 standard, there are some deviations/variations which prevent straightforward porting of IEC 61499 systems between tools.

We found that NxtStudio, compared to FBDK and 4DIAC, utilizes more complicated XML structure and attributes/element specific only to NxtStudio. When done manually, higher effort in modifying the FB source code file is needed when importing from FBDK or 4DIAC to NxtStudio. Also, another main portability issues between the tools are the graphical elements, which are less likely to be portable since these tools have their own ways of implementing graphical elements in the program (which is not defined by the IEC 61499 standard). Based on our findings, an IEC 61499 converter tool was created to automate the modification required by the FB source file and system configuration file, so that they will be accepted by different IEC 61499 tools.

References

[1] V. Vyatkin, "Software Engineering in Industrial Automation: State-of-the-Art Review," *IEEE Transactions on Industrial Informatics,* vol. 9, pp. 1234-1249, 2013.

[2] International Electrotechnical Commission, *International Standard IEC 61131-3: Programmable Controllers. Part 3: Programming Languages*: IEC, 2003.

[3] W. W. Dai and V. Vyatkin, "A case study on migration from IEC 61131 PLC to IEC 61499 function block control," in *2009 7th IEEE International Conference on Industrial Informatics*, 2009, pp. 79-84.

[4] International Electrotechnical Commission, "Function Blocks–Part 1: Architecture, International Electrotechnical Commission, Geneva," Switzerland, Technical Report IEC 61499-12005.

[5] V. Vyatkin, *IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design*, 3 ed. USA: Instrumentation Society of America, 2015.

[6] L. Ferrarini and C. Veber, "Control functions design and implementation of distributed automation systems for manufacturing applications," *International Journal of Manufacturing Research,* vol. 1, pp. 442-465, 2006.

[7] P. J. Brown, *Software Portability: an advanced course*: CUP Archive, 1979.

[8] C. Sunder, A. Zoitl, J. H. Christensen, V. Vyatkin, R. W. Brennan, A. Valentini*, et al.*, "Usability and Interoperability of IEC 61499 based distributed automation systems," in *2006 4th IEEE International Conference on Industrial Informatics*, 2006, pp. 31-37.

[9] J. H. Christensen, T. Strasser, A. Valentini, V. Vyatkin, A. Zoitl, J. Chouinard*, et al.*, "The IEC 61499 function block standard: Software tools and runtime platforms," *ISA Automation Week,* vol. 2012, 2012.

[10] T. Strasser, A. Zoitl, J. H. Christensen, and C. Sünder, "Design and Execution Issues in IEC 61499 Distributed Automation and Control Systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews),* vol. 41, pp. 41-51, 2011.

[11] C. Pang, S. Patil, C. 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)*, 2014, pp. 440-445.

[12] HOLOBLOC. (2018, 15 April). *FBDK 3- The Function Block Development Kit*. Available: http://www.holobloc.com/fbdk3/index.htm

[13] ISaGRAF. (2016, 9 March). *ICS Triplex ISaGRAF Inc.—leading IEC 61131 and IEC 61499 software*. Available: http://www.isagraf.com/pages/products/Isagraf/appworkbench_detail.htm

[14] 4DIAC. (2016, 9 March 2016). *4DIAC-IDE*. Available: http://www.eclipse.org/4diac/en_ide.php

[15] nxtControl. (2018, 15 April). *nxtControl - nxtStudio*. Available: http://www.nxtcontrol.com/en/engineering/

[16] 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)*, 2013, pp. 1-4.