IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 14, 2021, accepted December 10, 2021, date of publication December 22, 2021,
date of current version January 7, 2022.

Digital Object Identifier 10.1109/ACCESS.2021.3137642

Proposing and Prototyping an Extension to the Adapter Concept in the IEC 61499 Standard

PRANAY JHUNJHUNWALA[1], (Graduate Student Member, IEEE), AND VALERIY VYATKIN[1,2], (Fellow, IEEE)

[1]Department of Electrical Engineering and Automation, Aalto University, 02150 Espoo, Finland
[2]Department of Computer Science, Electrical and Space Engineering, Luleå Tekniska Universitet, 971 87 Luleå, Sweden

Corresponding author: Pranay Jhunjhunwala (pranay.jhunjhunwala@aalto.fi)

This work was supported in part by the HORIZON2020 Project 1- SWARM, and in part by the European Commission under Grant 871743.

ABSTRACT Component-Design Architecture has been in demand based on the growing needs for modularity and flexibility in the automation industry. IEC 61499 standard, a component-based automation architecture, provides various tools and techniques for automation developers to accommodate the need for flexibility in automation sequences. However, the adapter concept, one of the significant features of the standard, remains untouched and undeveloped since its inclusion in the standard and lacks the utilization of its true potential. In this work, we enhance the adapter concept by proposing the addition of logic into them. This proposition advances the adapter technology and gives the automation standard more capabilities to support higher levels of modularization without the increase of applications complexity.

INDEX TERMS IEC 61499, adapters, handshaking, message verification, sub-application, component-design.

I. INTRODUCTION

Industry 4.0 brings the need for flexibility in production scenarios in the automation industry. With the growing need for flexible production, the need for distributed and flexible automation has been highlighted. Distributed automation production scenarios replace large and costly controllers with various small controllers connected over the wireless networks. However, the need for distributed architecture and the requirement for flexibility, has revealed a gap in higher modularity standards in the industry.

A critical factor in achieving these higher modularity and flexibility standards is enabling cross-vendor product integration, defined as the seamless integration of devices produced and developed by different vendors. Providing such cross-vendor support compatibility is crucial at the physical level and the level of the automation architecture. Component design at the software level is necessary to facilitate these needs at the automation architecture level. Component design can be described as programming each part as individual components or a set of components, which encapsulate the implementation of the automation program. These components can be easily replaced, deployed and providing a set of interfaces for easy integration with other modules of the architecture is necessary.

The IEC 61499 standard is a component-based architecture providing the necessary means for automation system developers to work and develop applications that require modularity and flexibility. IEC 61499 has well-defined interfaces, which helps better component interactions. The standard also supports a visual component design approach that appeals to the automation systems developers in comparison to a purely textual programming language. The graphical programming method is more attractive for the developers because various components and modules are connected using connection links. Even though graphical, inter-component connections and interactions can sometimes be challenging because of various modules' complicated and extensive interfaces.

Adapter links, an integral component of the IEC 61499 standard, are an efficient solution to abstract out the complexity of inter-component relationships. Adapter links are used to simplify connections and communications between various modules of the automation program, making the integration and replacement of components more accessible and feasible. IEC 61499 is not only a component-based architecture but also supports distributed architecture. This means that components may be distributed across devices and may have to communicate via networks.

The associate editor coordinating the review of this manuscript and approving it for publication was Porfirio Tramontana.

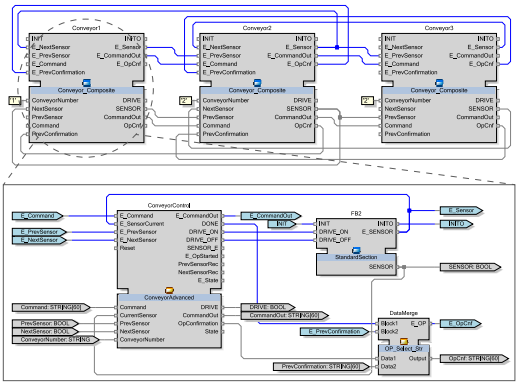

FIGURE 1. Composite function block network and internal composition.

Furthermore, the reliability of such connections may need to be ensured at the application level following the end-to-end principle, which can require complex protocols.

The existing adapter concept of IEC 61499 does not allow hiding this complexity in the adapters, which means the developers need to include additional modules to ensure this cross-device reliability which complicates the application design. This paper aims at addressing this issue by proposing an extension to the adapter mechanism allowing encapsulation of logic into the adapter interfaces. The benefits of the proposal are confirmed on a presented case study.

This paper is structured as follows: Section II introduces the IEC 61499 standard and explains the standard's essential parts. Section III highlights the previous work done using the IEC 61499 standard, development using adapters, and highlights the current limitations. Section IV explains the proposed extension to the adapter concept, followed by the used test-bed in Section V. In Section VI, the use-case is discussed. Finally, section VII explains the prototyping of the proposed extension in the IEC61499 standard, followed by the Discussion and Conclusions in Section VIII.

II. DISTRIBUTED COMPONENT AUTOMATION ARCHITECTURE OF THE IEC 61499 STANDARD

An extension to the IEC 61131-3 [1], standard for programmable logic controllers(PLC), IEC 61499 [2], is a component-based architecture that enhances the existing IEC 61131-3 by means of distributed systems and architecture.

A Function Block(FB) is the fundamental structural element of the IEC 61499 standard and can be of three kinds: basic, composite, or service-interface. A set of connected FB's have been shown in Figure 1, and the FB interface definition can be later seen in Figures 14a and 15.

IEC 61499 being a component-based architecture, the FB's have very well-defined interfaces that encapsulate event inputs and outputs, along with associated data inputs and outputs.

Basic FB's in the IEC 61499 are the building blocks of the automation program being developed. Besides the standard interface of a FB, they also support internal variables. Internal variables declared in the basic FB are not displayed on the interface of the basic FB, i.e., the internal variables are secure and cannot be modified from outside. They can only be modified during internal processing. Operation or processing by a basic FB depends on a state machine referred to as the Execution Control Chart(ECC).

ECC's in the IEC 61499 standard are similar to the Moore-Type state machine. ECC's can have numerous states connected to one another using transitions with guard conditions. Only when the guard condition is satisfied will the ECC transition from one state to another. Each state in the ECC can contain single or multiple actions. Actions are composed of two parts, an algorithm and an output event to be fired. Usage of the actions and the inclusion of algorithms or output events are all dependent on the requirements; a state could use both algorithm and event output or use either of them or use none.

IEC 61499 operates on an event-driven scenario and FB's are activated using event inputs, which are processed based on the applications logic, and event outputs can be generated based on the same. Various FB's, irrespective of their types, can be connected together using event and data connections, resulting in a 'Function Block Network' shown later in Figure 12. The execution order of the FB network is determined based on the event connections, the internal logic of the FB's, and also depends on the received event sequence.

Composite Function Blocks(CFB) type, in the standard, are used to combine a FB network into one large FB. CFB's can be composed of a network of only basic FB's or a combination of CFB's and basic FB's. As a result, programmers and engineers can use CFB's to develop more extensive hierarchical automation programs and applications. Shown in Figure 1 are three CFB's connected with one another, along with the CFB's internal composition.

The main goals of the IEC 61499 standard were to permit distributed deployment of FB's across various devices which has been further explained and demonstrated later in section V, in which the control program for the application has been distributed across 9 different controllers. The standard also incorporates additional communication FB's that can link the FB's in the network to external devices and controllers that do not operate on the IEC 61499 standard.

Another essential feature of the IEC 61499 standard is the Sub-application FB. Sub-applications are the same as CFB, with the only difference being that they allow deployment of their internal compositions to distributed devices compared to a regular CFB. Thus, the sub-application enhances the concept of CFB providing more flexibility to the application and developers. The sub-application technology has been further discussed in detail in section VII.

The standard also defines an adapter technology, which further enhances the interfaces and interactions between various FB's in the network. As shown in Figure 2, adapters were introduced to replace numerous event and data connections between various CFB's or Sub-Applications in the network by a single thick connection that would encapsulate both the events and data connections. Adapters encapsulate the

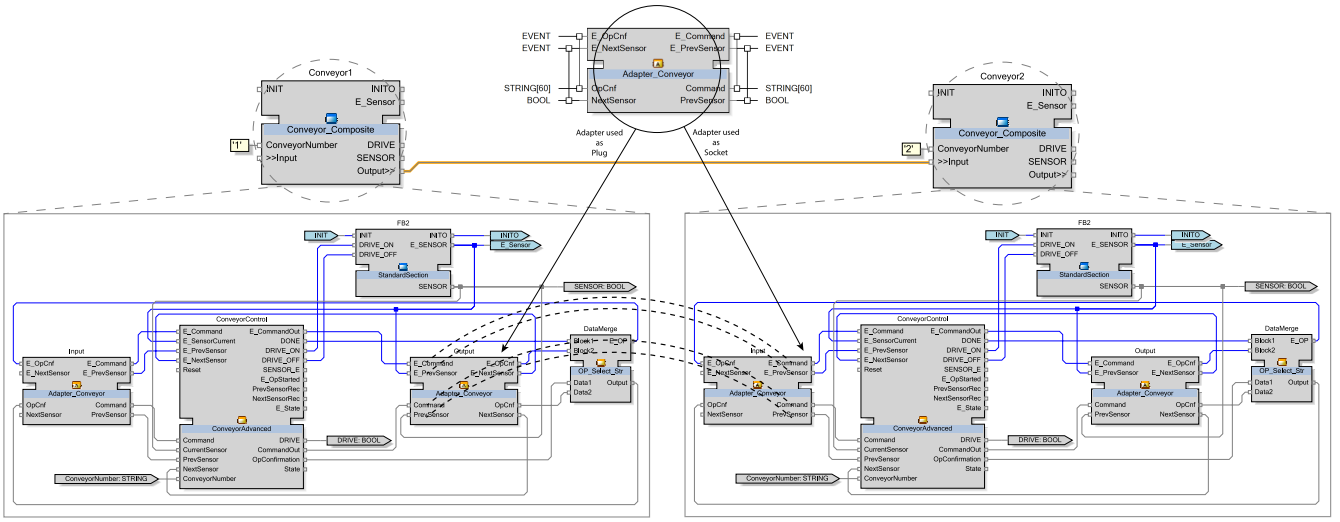

FIGURE 2. Adapter interface of IEC 61499.

connections and enable two-way communication between the FB's they connect.

Each adapter definition consists of two parts, i.e,. the 'Adapter Plug' and 'Adapter Socket'. As shown in Figure 2, plugs and sockets mirror each other's interface wherein plugs are defined at the output of a FB, whereas sockets are defined as the input of a FB. In Figure 2, we have implemented the adapter technology above the regular data and event connections between CFB's shown in Figure 1. As we can observe, event and data lines running across in both directions are encapsulated in the thick central orange connection making the network easier to access, debug and operate.

For more information on IEC 61499, we direct the reader to the proper introductory material, such as the book [3].

III. RELATED WORKS

IEC 61499 provides an appropriate engineering platform for adapting the object-orientation concept to industrial automation. A similar trend has been later addressed by the Asset Administration Shell (AAS) of the RAMI 4.0 architecture [4]. The main idea of the adaptation is to provide design structures for encapsulation of asset functionalities, masking their complexities in the applications where the assets are involved in. The relevant artifacts of IEC 61499 are basic function blocks with embedded state machines, CFB's and Sub-Applications, enabling the construction of hierarchical applications and adapters, encapsulating complex interconnections between components into one line. The related works date back to the concept of Automation Object [5], [6], which evolved to the concept of intelligent mechatronic components (IMC) [7].

Researchers always have focused on seamless communication between the software components representing assets, exemplified by works [8] and [9].

Several works have focused on enhancing the engineering process of component automation systems in the context of IEC 61499. Some early summaries can be found in [10], [11]. In particular, J.Christensen proposed using adapter interfaces for a tidier implementation of the MVCDA object-oriented design pattern in [12].

Zoitl *et al.* [13] present a method for developing modular, reusable IEC 61499 control applications in the 4DIAC IDE, and [14] explicitly focuses on hierarchical applications design, demonstrating the use of the adapter connections.

In [15], the authors set the one-line engineering design pattern based on the use of adapters. The authors explain the need for the addition of additional logic to ensure communication across devices when blocks of the one-line engineering application are deployed to different devices.

In a recent standardization work of OPAF [16] and [17], the adapter concept is widely used as a design artifact for complex process control applications.

Handshaking implementation with adapters was demonstrated in [18], and in [19], the authors provide a basic model of handshake message verification systems used to enhance communications reliability across smart devices and controllers. Kajola *et al.* [20] propose an extension to IEC 61499 to allow dynamic adapter connections, which allow for re-targeting plugs and sockets of interacting subsystems during runtime.

Dai *et al.*in [21], showcase the methodology of implementing service-oriented architecture based applications with adapters and highlight the easy replacement of components given that the adapter and service interfaces are the same and the use of adapters to implement plug-and-play of mechatronic components was demonstrated in [22].

In the recent surveys conducted by Lyu and Brennan in [23] and [24], it is highlighted that the IEC 61499 standard adopts the object-oriented programming pattern to design control applications and also mentioned that adapters provide a kind of inheritance similar for FB's to share standard interfaces. However, they lack the functionalities to attain the goals of

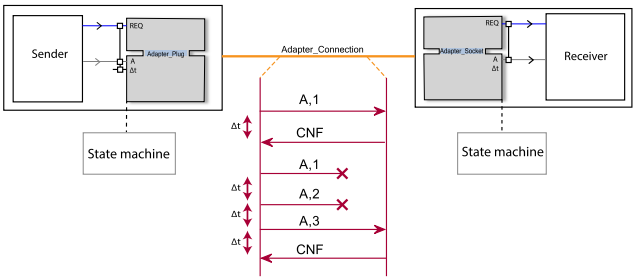

FIGURE 3. The extended adapter concept in a nutshell.

polymorphism and inheritance. The authors in the discussion in [24] emphasize the adapter design for IEC 61499 with computing paradigms and CPS in Industry 4.0.

In [25], the authors discuss the benefits of the microservices architecture for advanced manufacturing systems. According to Homay *et al.*higher standards of flexibility, modularity, heterogeneity can be achieved with the help of microservices. Furthermore, microservices help develop support for plug & play systems, one of the larger goals of Industry 4.0.

The cited works provide a convincing set of challenging use-cases for the adapter mechanism of component communication. In private communication of the authors with industrial adopters of IEC 61499 and in our own research work, it is evident that the adapter mechanism is an attractive instrument for application developers. On the other hand, when actively using it, the developers experience certain limitations which were not evident in the initial stages of the concept development.

IV. EXTENDED ADAPTER IDEA

In the extension of the adapter concept proposed in this paper, we propose ''embedding'' some logic into the standard adapters. The extended adapters will perform the standard adapter operation and, based on the included logic, perform some additional operations on the data before communicating them via the adapter connection. Finally, this modified data will be processed again based on the included logic at the receiving end, i.e., plug or socket.

The user/application-level interface of the extended adapters remains similar to the standard IEC 61499 adapters, as shown in Figure 3.

Similarly to the standard operation of IEC 61499 adapters, when the sender generates an event and data, data 'A' and event 'REQ' will be carried by the adapter connected to the receiving side where the adapter socket will split the event and data, and then pass it onto the receiver block.

In Figure 3, we demonstrate the working on an example of message retransmission in case of an unreliable connection. The plug and socket interface can have some additional elements to define the retransmission parameters, such as timeout duration or the limit on the number of retransmissions.

Upon the reception of a message from the sender, i.e., event input 'REQ' and the associated data input 'A,' the adapter plug before sending the data downstream processes it through the additionally added state machine, includes a message ID. It then transmits the data and event downstream to the adapter socket. It also enables an internal timer for Δt, within which it expects confirmation from the adapter socket. The user can manually set parameter Δt during the configuration of the extended adapters. If a confirmation is not received within the Δt period, the adapter plug state machine increments the count and resends the message using the same adapter link.

As shown in the sequence diagram in Figure 3, we first demonstrate the case when a confirmation, i.e., CNF event, is sent by the socket and is received within the Δt period. In the following case, we see a confirmation is not received for the first two transmissions; hence the plug increment's the value and resends the data.

When the bundled event and data are received via the adapter socket, it would be passed through the included state machine on the socket side, which will separate the message count from the message. The event 'REQ' and message 'A' will then be passed on to the receiver block downstream, and the socket will generate a CNF event which will be sent upstream to the plug, confirming the reception of the message.

The intended benefit of the proposed extension is in masking the complexity of complex communication logic by encapsulating it into the adapter connection.

The proposed notation of extended adapters will be explained in this section on a series of examples of increasing complexity.

A. SIMPLE RETRANSMISSION

Figure 4 illustrates the declaration of adapter interfaces implementing retransmission. The declaration is made for the pair plug-input and the socket-output. Similarly, the pair of socket-input and plug-output could be a subject of another such declaration. A textual syntax based on the standard has been included in Appendix A. Also a textual representation of adapters in Figure 4 has been highlighted in Appendix B.

However, as it will appear in subsequent use-cases, even the single pair declaration (plug-input, socket-output) could include additional signals for two other interfaces, i.e., plug-output and socket-input for processing requirements of the logic added on the plug-input and socket-output. As shown in Figure 4, the additional data declaration for the plug-input side consists of the interface variable `DT` of type `TIME`. It is needed to define the retransmission time.

Besides, two state machines (SM) are defined for the plug and socket side. It is assumed that the corresponding event and data of the adapters can be used in the respective state machines. The input elements can be used for reading and the output ones - for writing. Communication between the SMs is implemented using the `send` and `recv` commands. The notation is as follows: to send a variable A, say, from the plug-input SM to the socket-output SM, the command

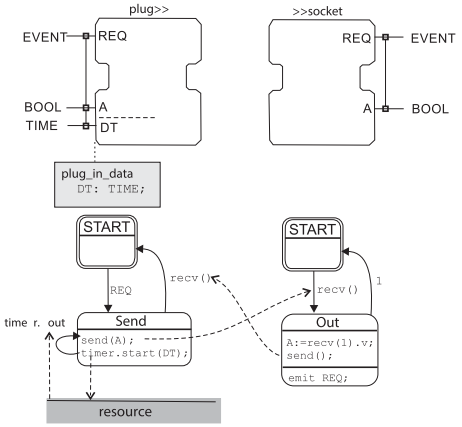

FIGURE 4. Declaration of adapter interfaces for retransmission.

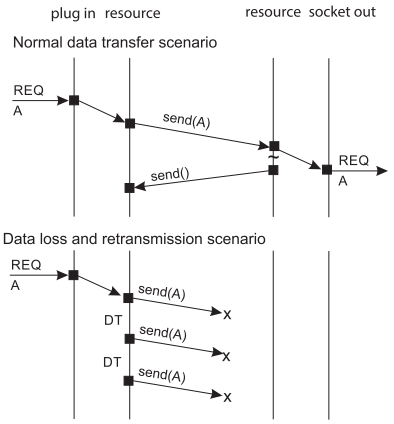

FIGURE 5. Normal data transfer and data loss scenarios.

`send(A)` is used. The command `send()` sends an empty message, effectively raising an event `recv()` on the other side. To access the message payload for non-empty messages, `recv(1).v` returns the value of the first element in the received message.

Figure 5 illustrates the behavior of the declared adapters when the input event REQ arrives.

The reader should note that the standard case of usual (non-extended) adapters can be represented using the introduced notation as shown in Figure 6. Here the event REQ and the associated data A and B are directly transferred from the plug side to the socket side without any additional transformations or actions. This implementation is assumed in the default case when the corresponding state machines are omitted in the adapter definition.

B. RETRANSMISSION WITH FINITE NUMBER OF ATTEMPTS

Figure 7 illustrates the declaration of adapters implementing a finite number of retransmission attempts, defined by another input `NR`. Counting the number of attempts required declaration of the integer variable `i` as an internal variable of the plug-input.

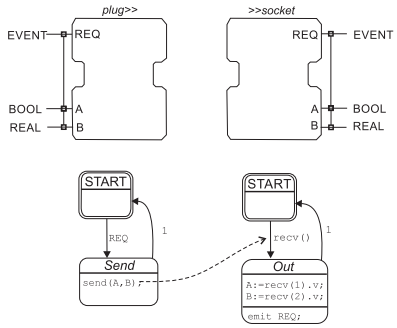

FIGURE 6. Implementation of the standard adapter using the notation of extended adapters.

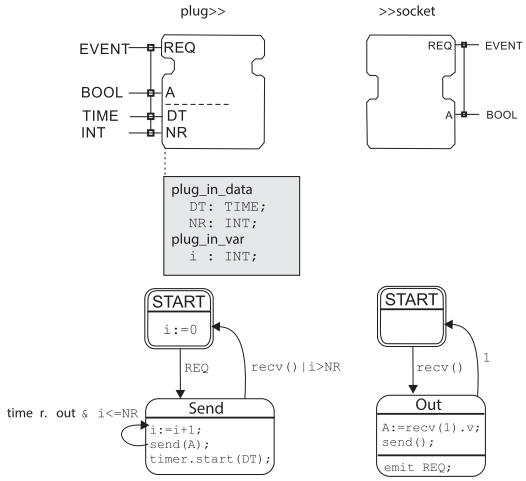

FIGURE 7. Adapter declaration for a finite number of retransmission attempts.

C. RETRANSMISSION WITH CONFIRMATION ON THE SENDER SIDE

Figure 8 illustrates the declaration of adapters implementing a finite number of retransmission attempts and producing a confirmation event `rsp` at the sender side. The confirmation event `rsp` will be transmitted to the blocks upstream, indicating that the sent data was successfully transmitted.

This required declaration of an auxiliary interface element: event `rsp` at the plug-output side, i.e., outside of the pair (plug-input, socket-output). We will use small letters for the auxiliary events to distinguish them from the events defined as a part of the main adapter interface.

D. DATA AND EVENTS ON THE RECEIVER's SIDE

Figure 9 illustrate the declaration of adapters which, in addition to the previously defined retransmission details, inform the receiving side about the number of retransmission attempts before it succeeded. This required declaration of the auxiliary data output `Nrt` at the socket output side. The message from the plug now includes both `A` and the counter `i`.

Figure 10 illustrates the behavior of the declared adapters when the input event REQ arrives.

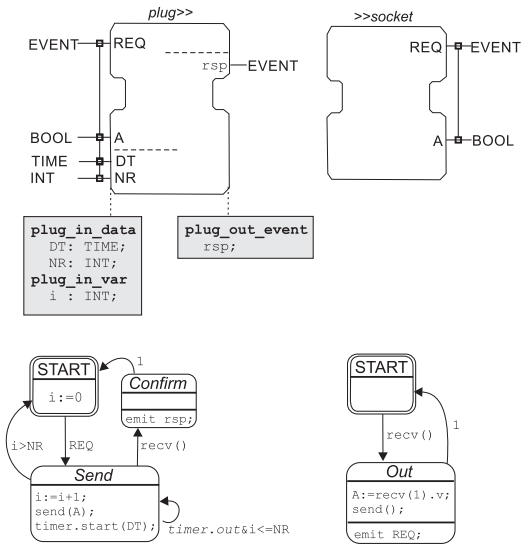

FIGURE 8. Adapter declaration for finite number of attempts with confirmation on the sender's side.

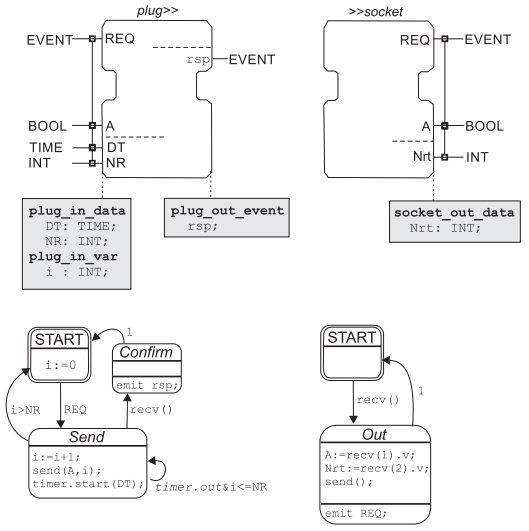

FIGURE 9. Adapter declaration for finite number of attempts with confirmation on the sender's side.

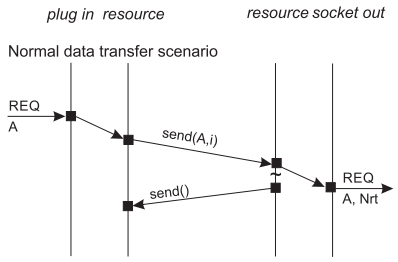

FIGURE 10. Normal data transfer scenario for the adapters from Figure 9.

E. EXTENDED ADAPTER TYPES DECLARATIONS

The extended adapter type declaration will require the following additional sections of interface and internal variables, for example, for the plug-input side:

1) `plug – input – event` - input events;
2) `plug – input – data`- input data;
3) `plug – input – var`-internal variables.

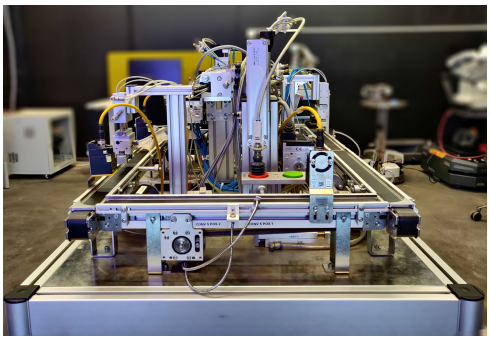

FIGURE 11. EnAS at the Aalto factory of the future[2](AFoF).

The ECC notation can be used to define the state machine logic of each of the four interfaces:

1) `plug – input – ECC`;
2) `socket – output – ECC`;
3) `socket – input – ECC`;
4) `plug – output – ECC`;

It is assumed that ECCs 1 and 2, and 3 and 4 can communicate with each other using the `send()`, `recv()` commands.

The ECCs can use the data declared for the same interface they belong to.

V. TEST BED-EnAS

Energy Autarkic Actuators and Sensors[1] (EnAS) is a testbed representing a small scale industrial production scenario and is used for the development and testing of various industrial automation techniques. Included with sets of pneumatic operators such as jacks and grippers, motor-driven conveyors, and laser sensors, EnAS, shown in Figure 11, provides researchers a platform to test their developments without the need for significant reconfigurability and hardware changes.

Shown in the upper right section of Figure 12 is the Top-View diagram of the EnAS demonstrator consisting of 6 motor-driven conveyors connected in a cyclic chain, a pair of pneumatic jacks, and grippers used to perform, pick and place operations over the work-pieces. Each conveyor is equipped with a laser sensor which is used to detect the position of the work-piece. Responsible for producing two spherical work-pieces, it can demonstrate and evaluate various automation techniques and scenarios.

The control application for the demonstrator has been developed using the NXTStudio software by NXTControl.[3] Figure 12 showcases the developed control application along with the device mapping schematic. EnAS is equipped with 9 controllers, which communicate with one other over the standard 2.4GHz WiFi protocol to facilitate and demonstrate distributed automation. Each hardware component has its own controller, i.e., each conveyor has its respective controller, and each pneumatic production island has its controller, i.e., Controller J1 and Controller J2. The 9th

[1]https://www.energieautark.com/
[2]https://www.aalto.fi/en/futurefactory
[3]https://www.nxtcontrol.com/en/engineering/

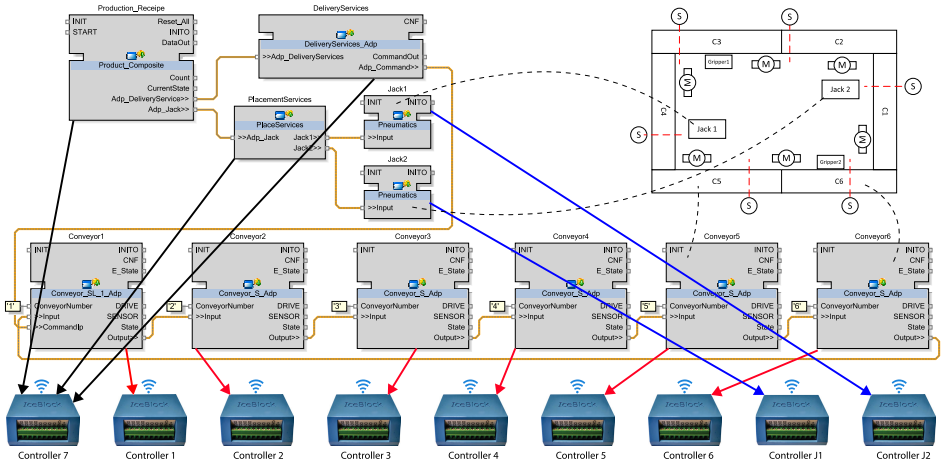

FIGURE 12. EnAS control application and distributed deployment.

controller, i.e., controller 7, is responsible for the top-level processes such as housing the HMI, the production scenarios, the delivery and placement services.

Across these devices, data either in `STRING` type or `BOOLEAN` type, or both are communicated. The control commands for the conveyors and jacks produced by the Production_Recipe FB and the operational confirmations by the low-level agents use the DataType `STRING`. The architecture of EnAS is a cyclic connection of conveyors. For sequential operations of the conveyors, each conveyor agent communicates its `Boolean` sensor reading to the conveyors connected upstream and downstream. The wireless distribution of the controllers controlling EnAS questions the reliability across these devices. Since these controllers communicate over simple 2.4GHz WiFi, packet and information loss has been a common point of failure reducing the reliability and success rate of the control application and production scenario. Therefore, a message verification system has been explained below in section VI to improve the reliability across various devices.

VI. USE CASE: HANDSHAKING AND RELIABILITY ACROSS DISTRIBUTED DEVICES

An advanced handshake message verification system has been developed to verify and exchange messages between two or more FB's. These FB's can be executed on the same device or distributed across various devices, but the main idea is to ensure reliable communication across distributed devices.

The handshake mechanism consists of two parts, i.e., the sender and the receiver, respectively. Illustrated in Figure 13 is the sequence diagram representing the operation of the handshake mechanism deployed between a controller(PLC) and motor. In Figure 13, we assume that the PLC and motor communicate using a wireless medium. In the desired case, i.e., Case 1, as soon as the handshake mechanism sender receives the command from the controller, it adds a message-ID and sends the message to the other side. Once

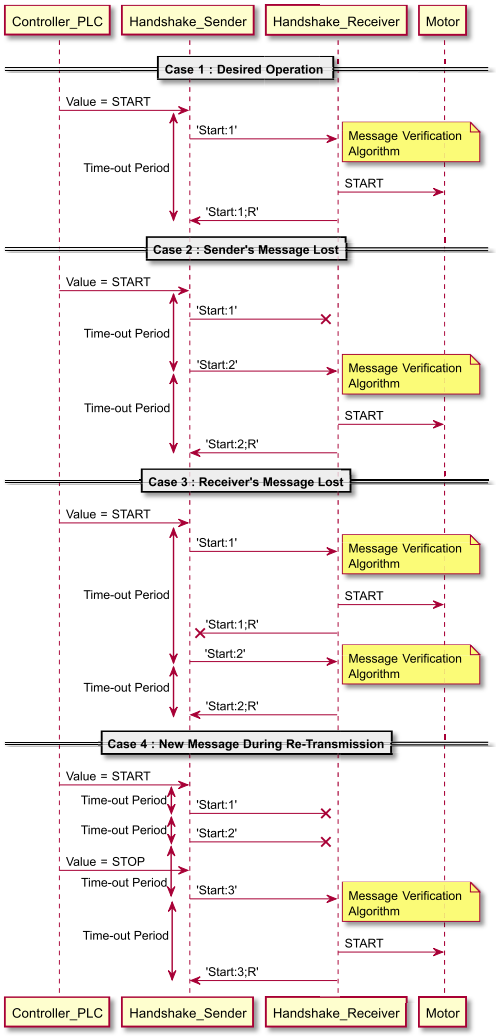

FIGURE 13. Handshake message verification operation chart.

received by the receiver, it would isolate the message and ID from one another and pass the message or command to the device downstream(motor in this case), following which it

will confirm to the sender by resending the received message along with the confirmation command ";R."

Since the communication takes place over a wireless channel, messages sent by the handshake sender or receiver can get lost in the case of a lossy channel. To ensure the successful delivery of the message to the receiver, the sender initiates a timeout period, in which it expects confirmation from the receiver after sending a message. If the confirmation is not received within the timeout period, the sender will resend/retransmit the original message but with an updated message-ID. The message-ID is incremented to inform the receiver that the command has been resent.

In Case 2 of Figure 13, we demonstrate the scenario in which the message sent is lost and does not reach the receiver. The receiver will not send a confirmation because it did not receive any message. Hence, when the timeout period elapses, the same command is again sent with an updated ID, i.e., '2', which is then processed by the message verification algorithm of the receiver.

In Case 3, we highlight the case where the confirmation message sent from the receiver is lost. Upon completion of the timeout period, the sender retransmits the message with an updated ID. The received message is then passed through the message verification algorithm of the receiver, and the receiver takes the desired action.

Cases 2 and 3 bring out the need for verification at the receiver's end. In its 'Message Verification Algorithm,' the receiver takes appropriate actions based on the command and message-ID. Implementations of this have been further explained in sub-section VI-A.

Case 4 in Figure 13 highlights a rather critical situation in which the controller sends a new message during the retransmission of an old message. When a new message arrives during the transmission of old commands or sequences, it is crucial for the message verification system to take into account the new message and ensure that both the old and new messages are transmitted to the receiving end. Case 4 has been handled at the sender's end because it is related to new messages received by the sender and is independent of the receiver's operation. Various SMs to counter-act this issue have been discussed below in sub-section VI-B.

A. HANDSHAKE MECHANISM RECEIVER

The receiver function block shown in Figure 14a has been designed to verify each incoming message from the sender based on the message, the message-ID, and the status of the previously executed operation. The message verification is done in the SMs 'MessageVerfication' state shown in Figure 14b. Initially, the verification algorithm isolates the message and ID into individual variables and checks if it is a new message, i.e., ID = 1. On the other hand, if the ID is 1, the SM proceeds as per regular operations shown in Case 1 of Figure 13. If the message ID is not 1, it is termed a retransmission, i.e., Cases 2 and 3 in Figure 13, for which the actions taken by the message verification algorithm have been explained in sections VI-A1 and VI-A2.

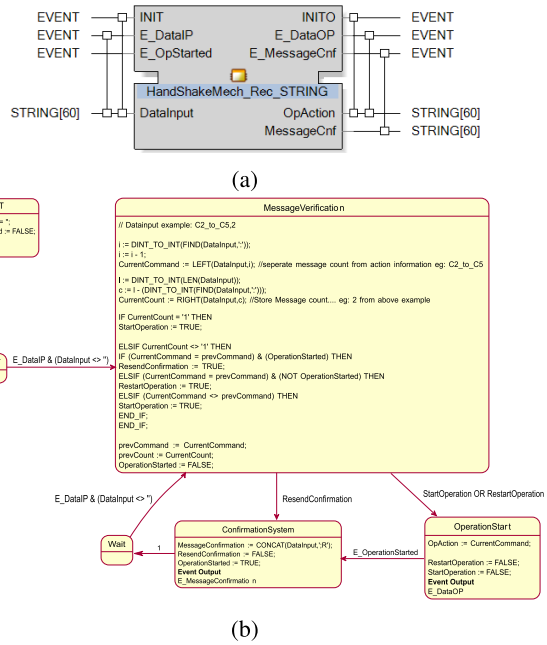

FIGURE 14. Handshake mechanism receiver a) Interface and b) ECC.

1) NEW COMMAND

In the case of a retransmission, i.e., message-ID > 1, the verification algorithm compares the received command to the previously received and started operation. If the command is new and was not previously processed by the receiver, it updates the message to the downstream blocks and then sends a confirmation to the sender, which is shown in Case 2 of Figure 13.

2) OLD COMMAND

When the received command is a retransmission of the previously processed the command, i.e., the case in which the command was passed downstream as highlighted in Case 3 of Figure 13, the receiver SM instead of going to the 'OperationStart' state directly jumps to the 'ConfirmationSystem' state in which it resends a new confirmation to the sender. Doing so, the receiver SM prevents the receiver from passing on repeated information to the blocks downstream.

B. HANDSHAKE MECHANISM SENDER

Shown in Figure 15 is the interface of sender FB, used for the handshake message verification system. The block receives as input the command to be transported and further attaches a message ID to the command before transmitting the message to the receiver. The event output 'EDelay1' and event input 'Delay1Done' are used to control the timer to check the Δt timeout period.

Case 4 of the retransmission shown in Figure 13 was tackled using 2 different approaches: 1) Checking for updated-values after each transmission 2) Checking for updated-values after N transmissions. Based on the performance of both approaches, the final SM for sender was deduced.

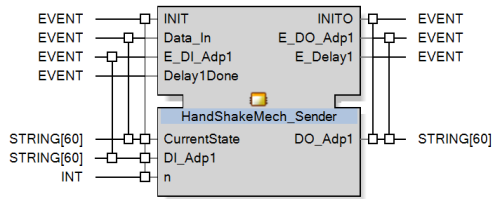

FIGURE 15. Handshake message sender function block.

TABLE 1. Handshake mechanism operational data.

	N	No. of Rounds	Errors	No. of Rounds/Error
SM 1	-	259	7	37
SM 2	2	365	9	41
SM 2	3	682	12	57
SM 2	5	386	9	43
SM 2	10	289	11	26
SM 3	3	1365	2	685

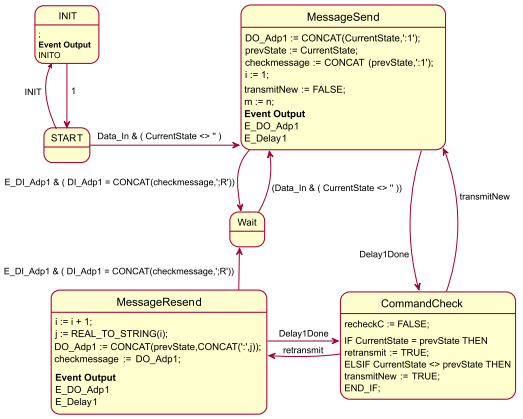

FIGURE 16. Handshake mechanism sender - SM 1.

To test and analyze the SMs, the number of production cycles completed before producing one error was calculated from the received data and analyzed. Then, based on the analysis of various modes of operation, the final state machines were designed and re-tested. Shown in Table 1 are the results for the various rounds of testings performed with the EnAS demonstrator.

1) CHECK UPDATED-VALUE AFTER EACH TRANSMISSION

The developed SM to check for updates after each transmission has been showcased in Figure 16. The SM re-checks the updated incoming command after each timeout period elapses. When an updated value is detected, the SM halts the retransmission and transmits the new updated value. However, if the value has not been updated the state machine resumes retransmission from the next message ID.

Using the SM shown in Figure 16, the system ran for 37 production rounds before giving an error and stopping the production.

2) CHECK UPDATED-VALUE AFTER N TRANSMISSIONS

Shown in Figure 17 is the SM developed to check for updated values after N retransmissions. The SM re-checks the updated

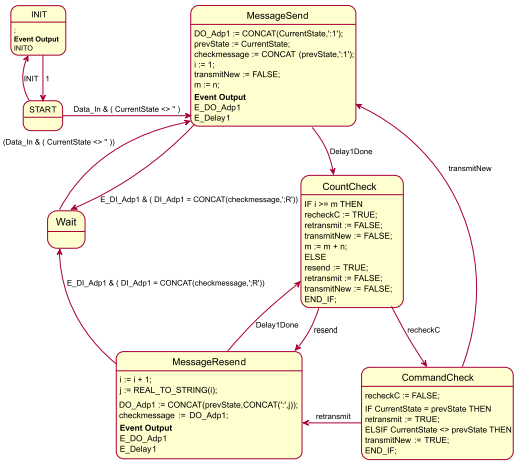

FIGURE 17. Handshake mechanism sender - SM 2.

incoming command after each 'N' retransmission of the previous command. When an updated value is detected, the SM halts the retransmission and transmits the new updated value. If the value has not been updated, the SM resumes retransmission from the following message ID and rechecks for updated values after N transmissions.

Various rounds of testing were performed by changing the number of retransmission 'N' and performing the production. Shown in Table 1 are the results from the testing, in which we observe with $N = 3$, the most number of rounds of production were completed before an error was received, and with $N = 10$, the efficiency was the least.

When N was 10, i.e., after retransmitting the message 10 times, the SM would check for an updated command. So, for example, if a new command was received during the 6^{th} transmission of the old command, and the receiver sends back a confirmation for the 6^{th} transmitted command, the retransmission would be halted, and then the sender block would wait for an updated command as discussed above.

Since in this case, N was 10, the sender SM would have checked for the updated command at the 10^{th} transmission. However, because the retransmission was halted at the 6^{th} transmission, the SM did not get into the state to re-check the updated command. Thereby missing the new command received during the 6^{th} transmission, resulting in a system error. Thus, when the value of N was reduced, the updated command was more frequently checked, resulting in the program completing more successful production rounds. For example, when N was 3, the SM at every 3^{rd} transmission would cross-check for updated value.

Therefore, according to the discussion above, when the value of N was reduced to 2, the efficiency should have increased. However, as we can see in Table 1, the number of successful rounds decreased, concluding that a certain number of retransmissions are needed for successful production rounds. In the case of the EnAS demonstrator the value was found out to be 3 retransmissions.

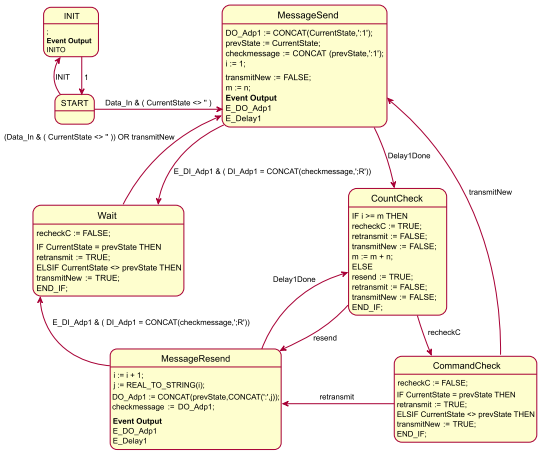

FIGURE 18. Handshake mechanism sender state machine.

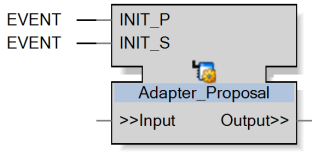

FIGURE 19. Sub-application function block interface.

3) SENDER STATE-MACHINE

Based on the analysis of SM1 and SM2 above, we concluded that the system needs a certain number of retransmissions of old-messages, but should also check for updated messages at a fixed interval. Furthermore, the analysis revealed the need to check for an updated value once a confirmation was received. This additional check needed to be performed irrespective of the retransmission count. Hence, a new SM shown in Figure 18 was developed and tested based on the analysis.

Instead of checking after each retransmission, the application would retransmit the old command N times. Other than the retransmission, the modified SM included an additional state that would check the input commands' updates after the sender confirmed the old command. The updated state machine, i.e., SM 3 resulted in the most efficient retransmission mechanism and produced only one error after 685 rounds. The final SM shown in Figure 18 was housed as the ECC for the Handshake sender FB represented in Figure 15.

In the following section VII, the handshake sender and receiver FB's and the respective SMs, will be used to develop sub-application FB using which the notion for the extended adapters will be demonstrated and proven.

VII. PROTOTYPING WITH SUB-APPLICATION

The idea behind the proposed extended adapters is to incorporate additional features or mechanisms into existing adapter connections used in applications. Since the current IEC 61499 tools do not support the proposed adapter extension; it was prototyped using the existing means.

Shown in Figure 19, is the developed sub-application FB used to demonstrate the proposed addition of logic to the adapter technology. The sub-application was used because of its capability to permit the deployment of various FBs in its composition to distributed devices. The developed sub-application FB was easily incorporated in the existing control application for EnAS shown in Figure 12, and the updated application can be seen in Figure 20. The development and operation of the prototype will be explained in depth by breaking the sub-application FB into layers.

The composition of the developed sub-application is shown in Figure 21. The FB contains a single input and output adapter connection, which can be connected across two FBs communicating with one another. Based on the proposal of the extended adapters above, the adapter type definition is the same for the input and output. Along with that, the sub-application also contains two initialization event inputs which are used to initialize the 'Plug and Socket FB' composing the sub-application. Individual initialization events have been included due to the need for deployment to distributed devices.

In this prototype, mapping of the CFB's inside the sub-application was performed based on the adapter extension proposal. The 'Handshake_Plug CFB' is deployed to Controller 3 along with the Conveyor 3 FB, because the 'Handshake_Plug CFB' contains the logic that has to be added into the extended adapter plug for the Conveyor 3 FB. Similarly, 'Handshake_Socket CFB', containing the logic for the extended adapter socket is deployed to controller 4 along with the Conveyor 4 block.

Figure 22, an extension of Figure 21, showcases the composition of the Handshake_Plug CFB and the Handshake_Socket CFB. The CFBs use the developed handshake sender and receiver blocks and houses the SMs explained previously in section VI.

The extension proposal and the prototype developments have been carried out taking into consideration the possibility of asymmetrical communication between 2 distributed devices. In Figure 22, along with the composition, we also showcase the asymmetrical communication taking place between Conveyors 3 and 4.

The asymmetrical operation using the prototyped FB has been explained in depth in subsections VII-A and VII-B, in which we take as an example the `STRING Control` commands and the `BOOLEAN` sensor values from the EnAS control program to demonstrate the versatility of the developed prototype and proposed extended adapters.

A. COMMUNICATION FROM CONVEYOR 3 TO CONVEYOR 4

To ensure reliability, `STRING Control` commands from Conveyor 3 are sent to Conveyor 4 via the handshake message verification system. Event and `STRING` type data generated from the Conveyor 3 FB are received at the `E_Data1` & `Data1` of the adapter input. Passed on further, these go through the 'Sender_String FB,' which communicates the command along with the desired message-ID.

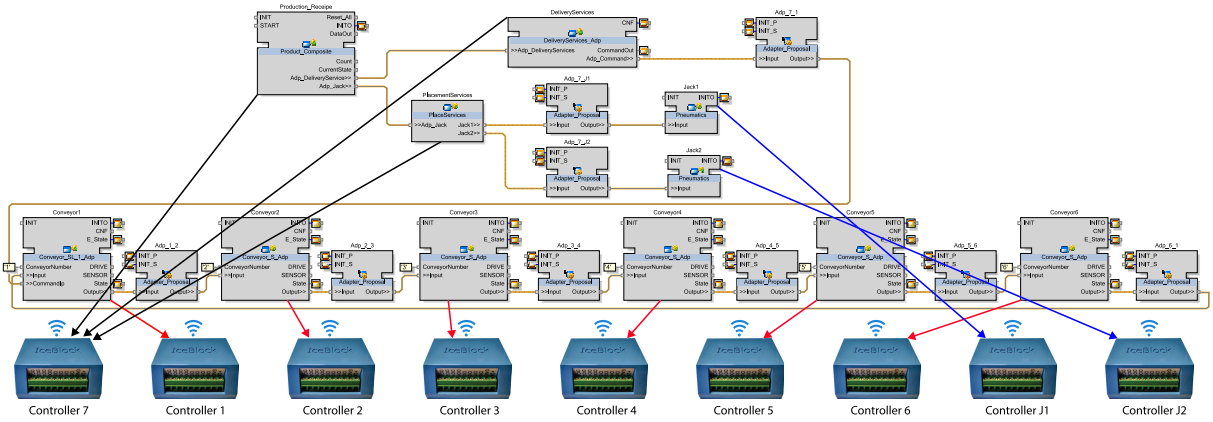

FIGURE 20. EnAS control application with sub-application.

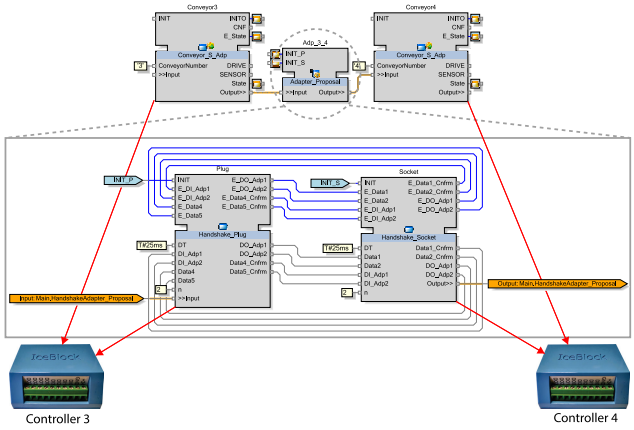

FIGURE 21. Sub-application composition and device deployment.

The sent message is received by Controller 4 at the 'Receive_String FB,' which then isolates the command and the message-ID from one another. The command is then forwarded to the adapter sockets' `E_Data1` & `Data1`, carrying the command to Conveyor 4. Based on the message-ID, the 'Receive_String FB', communicates the confirmation back to the 'Sender_String FB' inside the Plug in Controller 3.

B. COMMUNICATION FROM CONVEYOR 4 TO CONVEYOR 3

`BOOLEAN` sensor values from Conveyor 4 are sent to Conveyor 3 via the sub-application FB. The event and associated `BOOLEAN` sensor data generated from the Conveyor 4 sensor FB are received at the `E_Data5` & `Data5` of the adapter socket. Passed on further, these go through the 'Sender_Boolean FB,' which communicates the sensor value along with the associated message-ID.

Controller 3 receives the value at the 'Receive_Boolean FB,' isolating the sensor value and the message-ID from one another. The sensor value is then forwarded to the plugs' `E_Data5` & `Data5`, passing the sensor value to Conveyor 3 FB. Then, based on the message-ID, the 'Receive_Boolean FB', communicates the confirmation back to the 'Sender_Boolean FB'.

The ECC's in the prototype can be mapped to the extended adapter types declaration in section IV-E as follows:

1) `plug – input – ECC;` = Handshake_Sender;
2) `socket – output – ECC;` = Handshake_Receiver;
3) `socket – input – ECC;` = Handshake_Sender;
4) `plug – output – ECC;` = Handshake_Receiver;

ECC 1 and 2 communicate with one another, i.e., the case in section VII-A, and ECC 3 and 4 communicate with one another, i.e., the case in section VII-B. The same has been highlighted in Figure 22. Since the 'Handshake_Plug CFB' and 'Handshake_Socket CFB', are mapped to different devices, the mechanism ensures reliability across the wireless channel.

This concept and idea can be applied for N number of signals being transported in either of the directions. For each signal the system would communicate, a pair of sender and receiver handshake verification FB's would be required on each side of the adapter, i.e., plug and socket. For example, we need to communicate N commands from controller 3 to 4; we will need N number of handshake sender FB's on the adapter's plug side and N number of handshake receiver FB's on the socket side of the adapter. The same approach will be mirrored to enable two-way communication or transportation of data. There will be a need for N number of handshake sender FB's at the adapter's socket communicating with N number of handshake receiver FB's at the plug of the respective block.

VIII. DISCUSSION AND CONCLUSION

The intended contribution of this paper is to improve the design power of IEC 61499 by enhancing the adapter concept. This is achieved by masking the design complexity when implementing complex interactions between software components. The added functionality into the adapters aims to improve the standard's plug-and-play capabilities and semantic interoperability. It also improves reliability of distributed applications by reducing failure points. The proposal's overall impact is intended to reduce the design

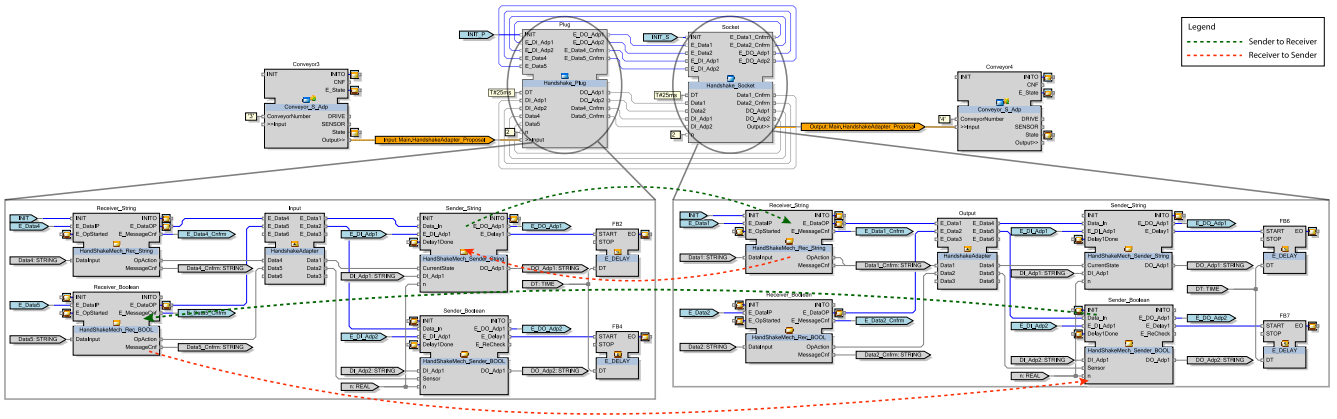

FIGURE 22. Composition of the Handshake_Plug and Handshake_Socket composite function block.

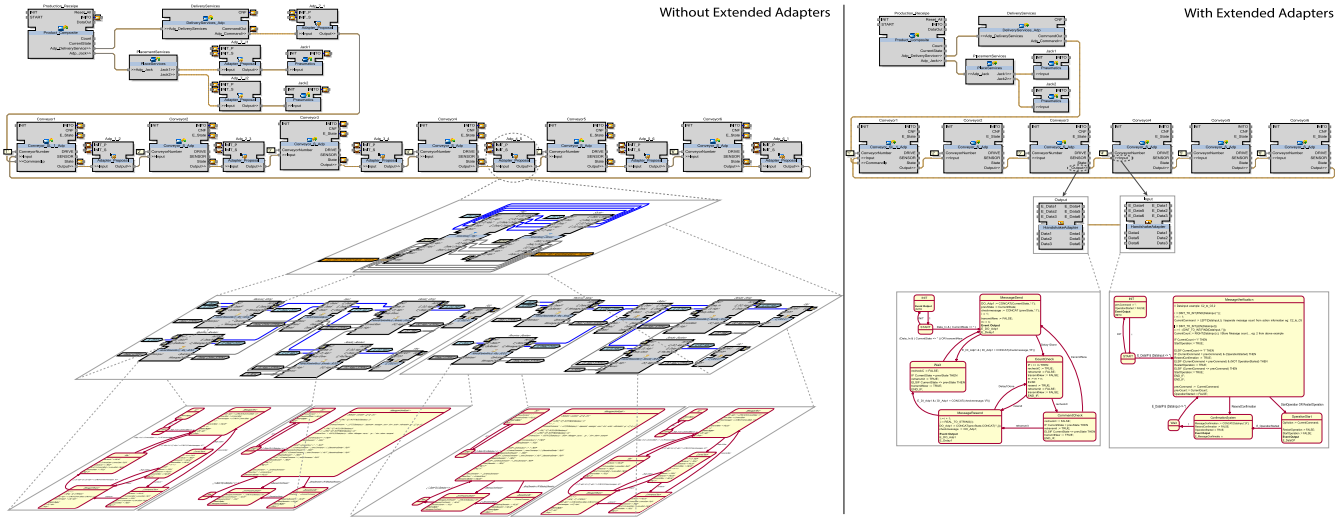

FIGURE 23. Application comparison showing the benefit of the extended adapters[4].

effort and improve the reliability of the resulting automation software.

The proposed extension was prototyped on example of a handshake message verification system used to ensure communication reliability across distributed devices. Since our concept is not yet implemented in the tools, it was prototyped using sub-applications. It was demonstrated that the developed sub-application FB was integrated into the same control application without much difficulty. Furthermore, the sub-application FB being an independent entity and separate from all aspects of the control program made debugging and testing of control sequences and low-level agents such as conveyors and jacks easier and more convenient for the developer. However, using the sub-application feature induced much additional complexity at the application level, as can be seen on the left side of Figure 23. To accomplish the goals of reliability across the distributed architecture numerous sub-application FB's had to be incorporated within the application. These sub-applications came with 3 levels of complex hierarchy and FB networks each. In addition, individual distributed mappings and deployment of these sub-application FBs had to be taken care of.

As discussed above in section VII, this approach could be scaled up to N number of signals being communicated between 2 devices, which means for each signal being communicated a pair of complex FB would have to be included into the control application, eventually making the application complicated and challenging to debug.

However, incorporating specific logic or operations into adapters could significantly reduce this added complexity. On the right side of Figure 23, we showcase the resulting structure of the application based on the proposed extended adapters. In the proposal, we have streamlined the use of state machines without the need of additional layers as can be seen by performing a side-by-side comparison of both applications. Additionally, we observe the reduced complexity at the application layer because of the extended adapter design. The inclusion of the state machines within

[4]The purpose of the figure is to compare the resulting application and show the proposal's benefit. Readers wanting improved readability are recommended to use the digital version and zoom-in.

the plug and socket interfaces would ensure that the adapters would still provide the basic functionality of the IEC 61499 adapters, along with which they would perform the operation of the added state machines if needed.

Furthermore, state machines included based on tested component re-use would ensure reliability due to the reduced failure points and also reduce debugging efforts for the engineers and developers.

Other than the stated advantages above, the authors believe that the microservice architecture applied in the IEC 61499 framework, as suggested by Homay *et al.*in [25] and Dai *et al.*in [26], would benefit from applying the adapter extension proposal of this paper. The expected benefits are in both engineering efficiency and reliability of runtime operation.

IX. FUTURE WORK

Future works would include the proposal of the extended adapters to the IEC61499 standard and the exploration of methods of integrating these adapters into software tools such as NXTStudio.

To standardize the developments, we plan to test the approach on other tests beds such as the FESTO CP-LAB[5] and on process-control applications on the test bed [27].

APPENDIX A
TEXTUAL SYNTAX OF PROPOSED ADAPTER EXTENSION

```
adapter_type_declaration : :=
      'ADAPTER' adapter_type_name
        fb_interface_list[6]
        [ad_internal_variable_list]
        [ad_ecc_declaration]
        [ad_algorithm_declaration]
      'END_ADAPTER'

  ad_internal_variable_list : :=
      'VAR'{internal_var_declaration}';'
        'END_VAR'

  ad_ecc_declaration : :=
        'EC_PLUG_STATES'
        {ec_states}[6]
      'END_PLUG_STATES'

      'EC_SOCKET_STATES'
        {ec_states}[6]
      'END_SOCKET_STATES'

      'EC_PLUG_TRANSITIONS'
        {ec_transition}[6]
      'END_PLUG_TRANSITIONS'

      'EC_SOCKET_TRANSITIONS'
        {ec_transition}[6]
      'END_SOCKET_TRANSITIONS'

  ad_algorithm_declaration : :=
      'ALGORITHM'algorithm_name'IN'
      language_type':'
        algorithm_body
      'END_ALGORITHM'

  algorithm_body : :=
      <as defined in compliant standards>
```

APPENDIX B
TEXTUAL REPRESENTATION OF PROPOSED ADAPTER EXTENSION

```
ADAPTER
    EVENT_INPUT
        REQ WITH A;
    END_INPUT

    VAR_INPUT
        A: BOOL;
    END_VAR

    VAR
        DT: TIME;
    END_VAR

    EC_PLUG_STATES
        START; (*Initial States*)
        Send : Send;
    END_PLUG_STATES

    EC_SOCKET_STATES
        START;
        Out : Out->REQ;
    END_SOCKET_STATES

    EC_PLUG_TRANSITIONS
        START TO Send := REQ;
        Send TO START := recv();
    END_PLUG_TRANSITIONS

    EC_SOCKET_TRANSITIONS
        START TO Out := recv();
        Out TO START := 1;
    END_SOCKET_TRANSITIONS

    ALGORITHM Send IN ST:
        send(A);
        timer.start(DT);
    END_ALGORITHM

    ALGORITHM Out in ST:
        A := recv(1).v;
        send();
    END_ALGORITHM
END_ADAPTER
```

[5]https://www.festo.com/us/en/e/technical-education/learning-systems/factory-automation-and-industry-4-0/learning-systems-industry-4-0/cp-lab-id_36133/

[6]Declaration is same as described in the Annex B.2.1 of the IEC 61499-1:2005(E) standard.

ACKNOWLEDGMENT

The authors would like to thank Hidenori Sawahara of Yokogawa Electric, TX, USA, for the motivation of this research.

REFERENCES

[1] *Programmable Controller—Part 3: Programming Languages*, Standard IEC 61131-3, Int. Electrotech. Commission, Geneva, Switzerland, 1993.
[2] *Function Blocks—Part 1: Architecture*, Standard IEC 61499, Int. Electrotech. Commission, Geneva, Switzerland, 2012.
[3] A. Zoitl and R. Lewis, *Modelling Control Systems Using*, document IEC 61499, IET, 2014, vol. 95.
[4] P. Adolphs, H. Bedenbender, M. Ehlich, and U. Epple, "Reference architecture model Industrie 4.0 (RAMI4. 0)," VDI/VDE, ZVEI, Tech. Rep., 2015.
[5] O. J. L. Orozco and J. L. Lastra, "Adding function blocks of IEC 61499 semantic description to automation objects," in *Proc. IEEE Conf. Emerg. Technol. Factory Autom.*, Sep. 2006, pp. 537–544.
[6] R. W. Brennan, L. Ferrarini, J. M. Lastra, and V. Vyatkin, "Automation objects: Enabling embedded intelligence in real-time mechatronic systems," *Int. J. Manuf. Res.*, vol. 1, no. 4, pp. 379–381, 2006.
[7] V. Vyatkin, "Intelligent mechatronic components: Control system engineering using an open distributed architecture," in *Proc. IEEE Conf. Emerg. Technol. Factory Automat. (EFTA)*, vol. 2, Sep. 2003, pp. 277–284.
[8] M. Hofmann and A. Zoitl, "Improved communication model for an IEC 61499 runtime environment," in *Proc. ETFA*, Sep. 2011, pp. 1–7.
[9] R. Froschauer, F. Auinger, A. Schimmel, and A. Zoitl, "Engineering of communication links with AADL in IEC 61499 automation and control systems," in *Proc. 7th IEEE Int. Conf. Ind. Informat.*, Jun. 2009, pp. 582–587.
[10] V. Vyatkin "IEC 61499 as enabler of distributed and intelligent automation: State-of-the-art review," *IEEE Trans. Ind. Informat.*, vol. 7, no. 4, pp. 768–781, Nov. 2011.
[11] V. Vyatkin, "Software engineering in industrial automation: State-of-the-art review," *IEEE Trans. Ind. Informat.*, vol. 9, no. 3, pp. 1234–1249, Aug. 2013.
[12] J. H. Christensen, "Design patterns for systems engineering with IEC 61499," in *Proc. Verteilte Automatisierung*. Magdeburg, Germany: Otto-vonGuericke-Universitaet, 2000, pp. 63–71.
[13] A. Zoitl, T. Strasser, and G. Ebenhofer, "Developing modular reusable IEC 61499 control applications with 4DIAC," in *Proc. 11th IEEE Int. Conf. Ind. Informat. (INDIN)*, Jul. 2013, pp. 358–363.
[14] A. Zoitl and H. Prähofer, "Guidelines and patterns for building hierarchical automation solutions in the IEC 61499 modeling language," *IEEE Trans. Ind. Informat.*, vol. 9, no. 4, pp. 2387–2396, Nov. 2013.
[15] G. Kollegger and A. Kopitar, "Flexible and reusable industrial control application," in *Distributed Control Applications*. Boca Raton, FL, USA: CRC Press, 2017, pp. 245–272.
[16] *Adapter Control*. Accessed: Jul. 1, 2021. [Online]. Available: https://www.holobloc.com/doc/fb/rt/opa/demo/Control.htm
[17] *Adapter Control*. Accessed: Jul. 1, 2021. [Online]. Available: https://www.holobloc.com/doc/fb/rt/opa/demo/Signal.htm
[18] S. Patil, D. Drozdov, and V. Vyatkin, "Adapting software design patterns to develop reusable IEC 61499 function block applications," in *Proc. IEEE 16th Int. Conf. Ind. Informat. (INDIN)*, Jul. 2018, pp. 725–732.
[19] P. Jhunjhunwala, U. D. Atmojo, and V. Vyatkin, "Towards implementation of interoperable smart sensor services in IEC 61499 for process automation," in *Proc. 25th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2020, pp. 1409–1412.
[20] P. Kajola, J. O. Blech, U. D. Atmojo, and V. Vyatkin, "Dynamic adapter connections for IEC 61499," in *Proc. 22nd IEEE Int. Conf. Ind. Technol. (ICIT)*, Mar. 2021, pp. 1054–1059.
[21] W. W. Dai, J. H. Christensen, V. Vyatkin, and V. Dubinin, "Function block implementation of service oriented architecture: Case study," in *Proc. 12th IEEE Int. Conf. Ind. Informat. (INDIN)*, Jul. 2014, pp. 112–117.
[22] R. Sinha, V. Vyatkin, Z. Salcic, and H. J. Park, "Competitors or cousins? Studying the parallels between distributed programming languages SystemJ and IEC61499," in *Proc. IEEE Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2014, pp. 1–7.
[23] G. Lyu and R. W. Brennan, "Towards IEC 61499 based distributed intelligent automation: Design and computing perspectives," in *Proc. IEEE 17th Int. Conf. Ind. Informat. (INDIN)*, Jul. 2019, pp. 160–163.
[24] G. Lyu and R. W. Brennan, "Towards IEC 61499-based distributed intelligent automation: A literature review," *IEEE Trans. Ind. Informat.*, vol. 17, no. 4, pp. 2295–2306, Apr. 2020.
[25] A. Homay, A. Zoitl, M. de Sousa, and M. Wollschlaeger, "A survey: Microservices architecture in advanced manufacturing systems," in *Proc. IEEE 17th Int. Conf. Ind. Informat. (INDIN)*, Jul. 2019, pp. 1165–1168.
[26] W. Dai, P. Wang, W. Sun, X. Wu, H. Zhang, V. Vyatkin, and G. Yang, "Semantic integration of plug-and-play software components for industrial edges based on microservices," *IEEE Access*, vol. 7, pp. 125882–125892, 2019.
[27] J. Peltola, J. Christensen, S. Sierla, and K. Koskinen, "A migration path to IEC 61499 for the batch process industry," in *Proc. 5th IEEE Int. Conf. Ind. Informat.*, vol. 2, Jun. 2007, pp. 811–816.

PRANAY JHUNJHUNWALA (Graduate Student Member, IEEE) received the B.Tech. degree in electronics and communication engineering from the Vellore Institute of Technology, Vellore, India, in 2019, and the M.Sc. degree in electrical and automation engineering from Aalto University, Finland, where he is currently pursuing the Ph.D. degree.

Since 2019, he has been a Research Assistant with the Information Technologies in Industrial Automation (ITiA) Group, Aalto University. He has also worked as a Teaching Assistant with the School of Electrical Engineering, helping students with various tasks and requirements of the courses. His research interests include distributed automation and industrial informatics, the IEC 61499 standard, software engineering for industrial automation systems, distributed architectures, and multi-agent systems.

VALERIY VYATKIN (Fellow, IEEE) received the Ph.D. and Dr.Sc. degrees in applied computer science from the Taganrog Radio Engineering Institute, Taganrog, Russia, in 1992 and 1999, respectively, the Dr.Eng. degree from the Nagoya Institute of Technology, Nagoya, Japan, in 1999, and the Habilitation degree from the Ministry of Science and Technology of Sachsen-Anhalt, in 2002.

He is on joint appointment as the Chair of dependable computations and communications with the Luleå University of Technology, Luleå, Sweden, and a Professor of information technology in automation with Aalto University, Finland. He is also the Co-Director of the International Research Laboratory Computer Technologies, ITMO University, Saint-Petersburg, Russia. Previously, he was a Visiting Scholar with Cambridge University, Cambridge, U.K., and had permanent appointments with the University of Auckland, New Zealand; Martin Luther University, Germany; and in Japan and Russia. His research interests include dependable distributed automation and industrial informatics, software engineering for industrial automation systems, artificial intelligence, distributed architectures, and multiagent systems in various industries: smart grid, material handling, building management systems, data centers, and reconfigurable manufacturing.

Dr. Vyatkin was awarded the Andrew P. Sage Award for the Best IEEE TRANSACTIONS paper, in 2012. He has been the Chair of IEEE IES Technical Committee on Industrial Informatics, from 2016 to 2019.