Integration of existing IEC 61131-3 systems in an IEC 61499 distributed solution

Stefano Campanelli, Pierfrancesco Foglia, Cosimo Antonio Prete
Dipartimento di Ingegneria dell'Informazione, Università di Pisa
Via Diotisalvi 2, 56126 Pisa, Italy
{stefano.campanelli, foglia, prete}@iet.unipi.it

Abstract

The IEC 61499 standard allows to model and design new generation control systems, providing innovative concepts of software engineering (such as abstraction, encapsulation, reuse) to the world of control engineering. The industrial reception of the standard, however, is still in an early stage, also because its introduction results in the adoption of a programming paradigm profoundly different than the widespread IEC 61131-3. This paper presents a method for the integration of the two standards, that allows to exploit the benefits of both. The proposed architecture is based on the parallel execution of both environments that interact with each other through some specific interfaces. A test implementation of the architecture is also presented to demonstrate the feasibility of the proposed solution.

1. Introduction

Programmable Logic Controllers (PLCs) are some of the most widespread devices used in industrial automation. In 1993, the International Electrotechnical Commission (IEC) published the IEC 61131-3 standard [1] in order to define a common programming interface for PLCs produced by different manufacturers. Since then, the standard has been widely adopted among PLC producers.

Evolution of computer networks brought the technology to realize control applications distributed between different devices. Distributed control is highly desirable in manufacturing industry, since it provides some benefits such as improved flexibility, reliability, maintainability and reduced wiring costs [2], however, a distributed control system is more difficult to design than a centralized system.

In order to facilitate the design of distributed control systems, IEC proposed the IEC 61499 standard [3] for distributed control and automation. The standard defines an open architecture that allows to model and design control applications where control logic is decentralized in software components that can be distributed across different hardware devices connected by networks [4].

The benefits of the standard have been proven through some practical case studies [5-7] and there are some commercial tools supporting the standard (ISaGRAF [8] and NxtControl [9]) already on the market. Despite this, the industry adoption of the standard is still in an early stage [10], this is due to challenges of both technical and economic nature that must be overcome to allow the widespread diffusion of the standard. Some of the technical challenges, such as predictability and scalability, are already discussed in [11] and [4]. From an economic point of view, companies have made investments in IEC 61131-3 and their interest is to preserve their know-how and developed software [12]. In fact, since IEC 61131-3 has been in use for years, there is a large number of control systems and software libraries that are based on this standard. Unfortunately, the IEC 61499 standard is very different from IEC 61131-3 in many ways, such as execution model or data handling, so it is not possible to directly port IEC 61131-3 applications in an IEC 61499 based runtime environment. In addition, IEC 61131-3 was quickly adopted by industry because its languages were very similar to the already existing non-standard languages used by control engineers to program PLCs, so personnel and design methodologies could be easily reused. On the contrary, IEC 61499 requires a different approach to the programming of control systems that makes it necessary to train personnel for the new standard. Moreover, existing design methodologies, patterns, competences and know-how acquired for IEC 61131-3 are not directly applicable in the design of IEC 61499 systems.

For the reasons specified above, it is not convenient for companies to convert existing systems to IEC 61499, nevertheless, many of these systems may take advantage from distribution of control logic to increase interoperability, reduce human work, improve control decisions, etc.

The first problem addressed in this paper regards the implementation of distributed control logic on existing IEC 61131-3 systems. As it has already been said, distributed control brings benefits to manufacturing industry, however, the IEC 61131-3 focuses on programming aspects of a single device. Some communication functionalities that can be used to realize

distributed control are defined in the IEC 61131-5 standard [13], but it lacks the modeling and abstraction aspects that have been introduced in IEC 61499 to simplify the distribution of control logic and the communication between devices. So, the implementation of a distributed control action between devices that supports only IEC 61131-3 and IEC 61131-5 would be a complex task.

On the other hand, IEC 61499 facilitates the design of distributed control logic, but its introduction in an already existing IEC 61131-3 system poses a new problem: the reuse of the existing IEC 61131-3 code in an IEC 61499 system. It has already been said that there are differences between the two standards that prevent the direct reuse of IEC 61131-3 applications in a IEC 61499 system. So, in order to reuse existing code, it is necessary to either port the existing code into IEC 61499 code or create an environment where both IEC 61499 and IEC 61131-3 can coexist. Harmonization of both standards can be achieved in different ways that are discussed in [12].

The last problem addressed in this paper is the reuse of acquired know-how and personnel competences about IEC 61131-3 in the design of a new IEC 61499 based system. In fact, the differences in execution and data handling make it impossible to reuse in IEC 61499 the same designing methods used for IEC 61131-3, so personnel should be trained for the new standard and all the existing competences and know-how would be wasted.

In this paper, we propose a method to realize the integration of IEC 61131-3 and IEC 61499 runtime environments with the aim of solving the three problems presented above:

1. realization of distributed control between existing IEC 61131-3 systems;
2. reuse of existing IEC 61131-3 software in an IEC 61499 system;
3. reuse of existing know-how and personnel competences about IEC 61131-3.

In the vision of the proposed solution, each physical device is able to execute both IEC 61131-3 and IEC 61499 code since it has a runtime environment for each standard. The proposed method consists of a software layer that realizes information exchange between the two runtime environments. The functionalities of this software layer are exposed to each runtime as constructs proper of each standard (in particular, IEC 61131-3 runtime sees the functionalities as IEC 61131-5 communication function blocks and IEC 61499 sees the functionalities as Service Interface Function Blocks).

The proposed method allows to overcome the three problems described above. The first problem is solved utilizing IEC 61499 to design the distributed control logic. Local information necessary to take distributed decisions can be provided to IEC 61499 applications by IEC 61131-3 tasks using the proposed interface. The second problem is solved because the existing code will continue to work, since the IEC 61131-3 runtime will still be present. Regarding the reuse of previous know-how, the new system will be developed in part using IEC 61131-3 and in part using IEC 61499. So, even though personnel trained in the IEC 61499 standard will still be required in order to design the distributed part of the system, the competences and know-how about IEC 61131-3 will not be wasted. In this way, an expert control engineer can choose the most suitable standard for the realization of different functionalities of the control system.

The rest of the paper is structured as follows: section 2 presents some related works, basic concepts of IEC 61131-3 and 61499 and their differences are discussed in section 3, the proposed integration method is described in section 4, in section 5 a basic implementation of the architecture is presented and the last section concludes the paper.

2. Related work

In this section some of the related academic work is presented; in particular we present first works aimed at the migration of existing IEC 61131-3 software to IEC 61499 platform and then some works aimed at the coexistence of the standards.

Migration case studies are presented by Gerber et al. in [14] and Dai and Vyatkin in [15], where some real world IEC 61131-3 systems are re-designed in IEC 61149 systems. While the first one focuses on the translation of each IEC 61131-3 code module into an IEC 61499 code module, the second one describes two different approaches to the redesign of the entire system. Both papers also give some general guidelines to simplify future migration processes. Similarly, Sünder et al. present in [16] some transformation rules to manually migrate a system.

Even though the recommendations, guidelines and transformation rules can facilitate the migration process, it still remains an expensive task. So, some automated approaches have been studied. Shaw et al. [17] presented an automated process to convert programs in ladder diagram into C programs to be included into IEC 61499 reusable code modules, while Wenger et al. [18] have automated the transformation of an entire IEC 61131-3 system following most of the rules presented in [16]. However, these approaches are incomplete and they are still not sufficiently compatible to convert a system without further human work to finish the migration process. Moreover, the code automatically generated has poor structure and it is not very readable, so it would be difficult to maintain.

On the other hand, Zoitl et al. [12] discussed about the importance of the coexistence and harmonization of both standards, seeing IEC 61499 more as a complementary standard to IEC 61131-3, rather than a

replacement. They proposed three different approaches to achieve harmonization between the two standards:

1. parallel use of both standards with a communication interface to provide interoperability;
2. IEC 61131-3 based system enhanced with IEC 61499 concepts;
3. IEC 61499 based system enhanced with IEC 61131-3 concepts.

To the authors' best knowledge, there is not much research or commercial work based on the first approach other than a proposed annex [19] that defines an interface that allows IEC 61499 to utilize some functionalities to write and read some remote variables and status information on IEC 61131-5 compliant devices.

ISaGRAF [8] is a commercial solution that utilizes the second approach, implementing the IEC 61499 concepts in an IEC 61131-3 environment. However, this implementation has some differences from the other implementations of the standard that are analyzed by Vyatkin and Chouinard in [20].

Sünder et al. [21] discussed the potentiality of a component based implementation of the third approach.

The proposed solution follows the first approach, improving the interaction possibilities between the two standards coexisting in each device, since information exchange occurs through a specific interface.

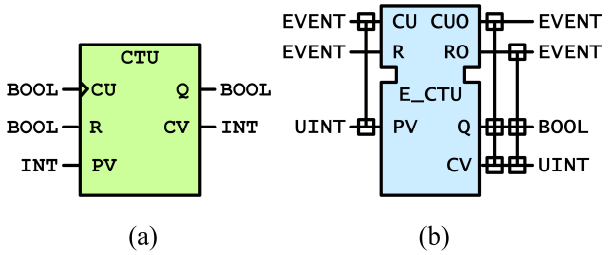
3. Basic concepts

In this section we present some basic concepts of IEC 61131-3 and IEC 61499 as long as the challenges to the integration of both standards.

3.1. IEC 61131-3 basics

The IEC 61131-3 standard defines a set of programming languages (some graphical and some text-based) as well as some common elements to all programming languages.

The common elements comprehend definition of data types, variables, Program Organization Units (POUs) and a software model of the PLC. This model is used to organize the execution of the program code into a set of tasks that can be of three types: cyclic (executed cyclically), periodic (executed at regular time intervals) or event (executed at each occurrence of a particular event). The POUs allow to structure the code into programs, functions and function blocks. In particular, the Function Block (FB) is a code module that can have multiple inputs, multiple outputs and some internal memory that maintains the status of the function block instance between successive invocations. Representation of an IEC 61131-3 FB is shown in Fig. 1 (a).



(a) (b)

Figure 1. Example of a IEC 61131-3 function block (a) and a IEC 61499 function block (b).

3.2. IEC 61499 basics

The IEC 61499 standard defines an architecture where a system consists of a set of devices connected by communication networks. The basic unit of reusable code defined by the standard is the function block. IEC 61499 function blocks extends the concept of FB defined in IEC 61131-3 adding event inputs and outputs in addition to data inputs/outputs. In Fig. 1 (b) an example of IEC 61499 FB is shown. Connections between event inputs and data inputs indicate which input data lines must be sampled at occurrence of a specific event. In a similar way, connections between event outputs and data outputs indicate which output data lines must be updated when a particular event is sent.

The IEC 61499 standard uses the same data types defined by IEC 61131-3.

The standard defines three kind of function blocks:

- Basic Function Block: its behavior is defined by a state-chart machine called Execution Control Chart (ECC) that is invoked each time an input event is received. The ECC triggers the execution of algorithms that may be written in any language (IEC 61131-3 languages are advised) and sends output events.
- Composite Function Block: it is composed by interconnecting other FBs.
- Service Interface Function Block: it realizes the interaction with the hardware or the operating system functionalities (such as communication or user interface). It must be provided by the manufacturer and its implementation is hidden, however its interface may be described using time-sequence diagrams.

Function blocks provide the concept of data and control abstraction, exposing an interface that hides implementation details of control logic. In such way, control systems can be modelled and prototyped without considering hardware or other implementation-specific details. Applications are networks of interconnected function blocks that may be distributed to different devices across the system.

3.3. Challenges to integration

The different execution model represents a challenge to the reuse of IEC 61131-3 code into an IEC 61499 FB.

IEC 61131-3 FBs are invoked when their task is executed, while IEC 61499 FBs are invoked when an

input event is received. Even though IEC 61131-3 allows to create event-driven tasks, the most used are the cyclic and periodic tasks. So, the execution method of IEC 61131-3 is cyclic, while IEC 61499 adopts an event-driven approach.

While the first method is more predictable, so it is more suitable for real-time applications, the second method allows better performance, since it invokes function blocks only when needed.

Even if IEC 61499 basic FBs allow to trigger the execution of IEC 61131-3 algorithms at the occurrence of events, many of the IEC 61131-3 FBs (such as the ones of IEC 61131-5) are designed for a cyclic use and are not suitable for single executions of programs.

Another challenge is represented by global data handling. IEC 61131-3 allows the creation of global variables shared by different POUs that may be executed by different tasks. Such a programming approach is widespread in industrial automation. In IEC 61499, however, interactions between FBs only occur through their interface. This improves portability, maintainability and allows to easily distribute code across devices. Unfortunately, this difference makes very difficult to reuse IEC 61131-3 FBs in a IEC 61499 runtime environment with no support for global data.

4. Proposed architecture

In this section the architecture of a distributed control system that allows to execute natively both IEC 61131-3 and IEC 61499 code is presented and the communication interface between the two standards is described.

4.1. General architecture

Realizing a tight coupling between IEC 61131-3 and IEC 61499 standards in a single execution environment without sacrificing some aspects of the standards is an hard task, maybe impossible, for the motivations presented in section 3.3. So, in order to remain strictly compliant to both standards, the proposed architecture is based on a loosely coupled approach, where each device can execute both IEC 61131-3 code and IEC 61499 code using different execution environments that have the possibility to interact through a communication interface. The proposed architecture is shown in Fig. 2.

Interactions between code of different standards must occur through a set of *PLC data exchanges* (PDEs) that are defined by the system designer. PDEs can be of two types:

- Data transfer PDEs: data are sent from a function block of one standard to a function block of the other standard;
- Procedure call PDEs: data are sent similarly to the precedent case, but also response data are expected.

The PDE is defined specifying an identifier, the PDE type (data transfer or procedure call), the parameters (and their data types) that have to be transferred and the direction (IEC 61131-3 to IEC 61499 or vice versa). PDEs are logically grouped in one or more *PLC interfaces* (PIs)

From the IEC 61499 side, the PLC interface is represented as a Service Interface Function Block (SIFB). Event and data inputs/outputs of the SIFB depend on the PDEs that the PI implements. For example, the SIFB shown in Fig. 3 (a) comprehends three PDEs: a procedure call PDE from IEC 61131-3 to IEC 61499 and two data transfer PDEs, one for each direction. PDEs are realized by triggering and/or receiving events of the following types:

- REQ: data exchange request;
- CNF: data exchange confirmation;
- RESET: reset request;
- IND: data exchange indication;
- RSP: data exchange response.

Each event input/output of the SIFB is associated to a single PDE, so if a particular kind of event is required by more than one PDE, it must be replicated (as shown in Fig. 3 (a) where the IND event is replicated). Connections between event and data input/outputs indicates the parameters to be exchanged. The general idea is to abstract an IEC 61131-3 device (or one of its functionalities) using a PLC interface that hides the actual IEC 61131-3 program and only shows the

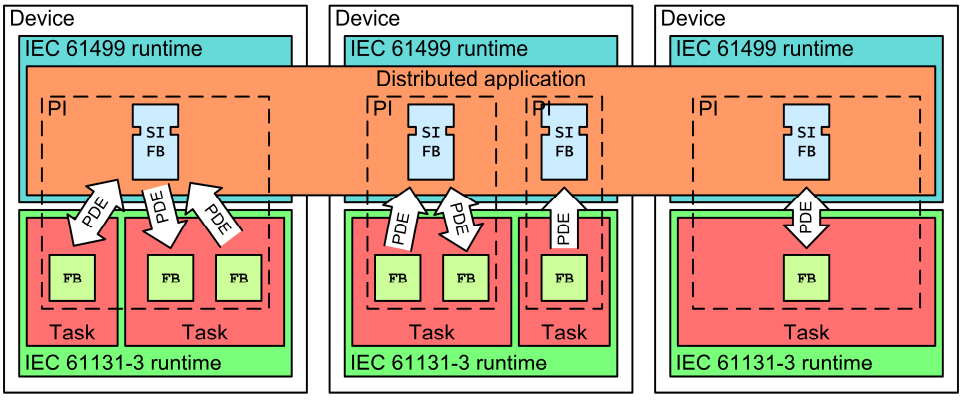


Figure 2. The proposed architecture.

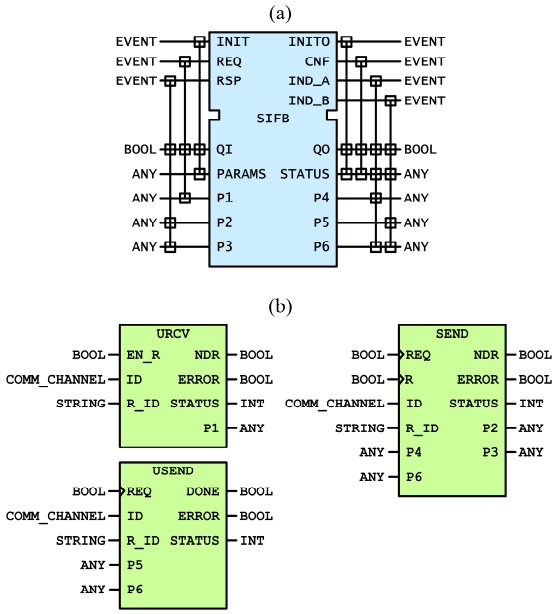


Figure 3. Example of a PLC interface: (a) IEC 61149 representation; (b) IEC 61131-3 representation.

interface as a SIFB.

From the IEC 61131-3 side, the whole PLC interface cannot be represented by a single function block instance, since different data exchanges can occur in different programs executed by different tasks. So, each PDE of the PI is performed by a different function block instance. Instead of defining a custom function block interface for data exchanges, the interfaces of the communication function blocks defined in the standard IEC 61131-5 are adopted. The PLC interface is modelled as an IEC 61131-5 communication channel, while PDEs are represented as services offered by the communication channel. In order to perform PDEs, instances of IEC 61131-5 communication function blocks are used. Fig. 3 (b) shows the IEC 61131-5 FBs used to realize the three PDEs described above.

4.2. Initialization of the PLC interface

Before the communications between standards through a PLC interface can start, the PI must be initialized in order to allocate resources necessary for the data exchanges. The initialization procedure must complete only when both sides correctly perform resource allocation and are ready to communicate.

From the IEC 61131-3 side, the initialization is performed by an IEC 61131-5 CONNECT function block (shown in Fig. 4) when the connection is enabled setting EN_C input to 1. The PARTNER input is used to indicate the identifier of the PI. When initialization is finished from both sides, the VALID output is set and the ID output will contain the channel descriptor. ERROR and STATUS outputs are used for error handling.

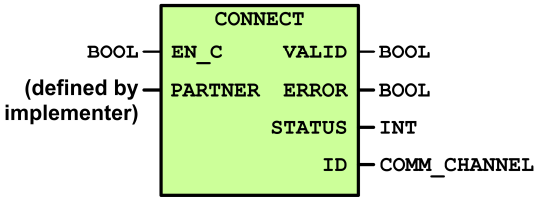


Figure 4. The IEC 61131-5 CONNECT FB.

From the IEC 61499 side, the resources are allocated during the initialization of the SIFB (already shown in Fig. 3 (a)). According to the standard, the initialization is started on reception of the INIT event with the input qualifier (QI) set to true. When the initialization is correctly finished (from both sides), the output event INITO is sent with output qualifier (QO) set to true.

4.3. Description of PDEs

In this section we explain how to realize each kind of PDE from both IEC 61131-3 and IEC 61499 perspective. The function blocks that allow to perform PDEs and the time-sequence diagrams that describe their semantics are shown in Fig. 5-8. The first examined PDE is the Data Transfer PDE from IEC 61131-3 to IEC 61499 (Fig. 5). This PDE allows IEC 61131-3 programs to send data to IEC 61499 applications. In order to accomplish this, IEC 61131-3 code must contain an instance of the IEC 61131-5 USEND function block, shown in Fig. 5 (a), that is used to communicate data to an instance of the SIFB associated to the PI contained in the IEC 61499 application, shown in Fig. 5 (b). As shown in the time-sequence diagram in Fig. 5 (c), when a rising edge is detected on the REQ input of the USEND FB, the PDE data (input parameters SD_1, … SD_n) are transferred to the SIFB (output parameters RD_1, … RD_n). The SIFB then trigger an IND+ event in order to indicate the data reception to the IEC 61499 application. The USEND FB signals that data is sent pulsing the DONE output.

The Data Transfer PDE from IEC 61499 to IEC 61131-3 (Fig. 6) allows to IEC 61499 applications to communicate data to IEC 61131-3 programs. An instance of the IEC 61131-5 URCV FB (Fig. 6 (a)) is used to receive data from the IEC 61499 SIFB (Fig. 6 (b)). According to the IEC 61131-5 standard, the URCV FB instance must be cyclically invoked to check if new data are present. It can be enabled or disabled using the EN_R input. Fig 6 (c) shows that the data transfer starts when a REQ+ event is received by the SIFB. If the URCV FB is enabled (EN_R input set to 1) data are transferred from the SD_1, … SD_n inputs of the SIFB to the RD_1, … RD_n outputs of the URCV FB and the NDR output of the URCV pulses to indicate the presence of new data. The SIFB can send a CNF+ event to indicate that the data transfer succeeded or it can send a CNF- event if the URCV is disabled to signal that the data transfer failed.

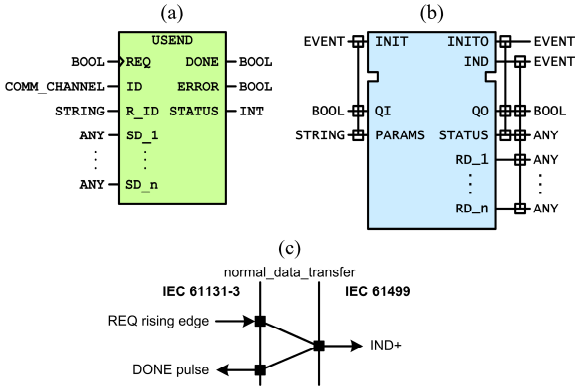


Figure 5. Data transfer PDE from IEC 61131-3 to IEC 61499: (a) IEC 61131-3 FB; (b) IEC 61499 SIFB; (c) time-sequence diagram.

Procedure Call PDEs allow to send data from a standard to the other one and to receive some data in response realizing a sort of inter-standard procedure call.

In order to perform a Procedure Call PDE from IEC 61131-3 to IEC 61499 (Fig. 7), an IEC 61131-3 program must contain an instance of the IEC 61131-5 SEND FB, shown in Fig. 7 (a), that is used to send data to the IEC 61499 SIFB shown in Fig. 7 (b). The SIFB signals the reception of data to the IEC 61499 application that uses them to compute response data that the SIFB sends back to the SEND FB, completing the PDE.

As shown in Fig. 7 (c), the procedure call starts on rising edge of REQ input of the SEND FB that transfer data from its SD_1, … SD_n inputs to the RD_1, … RD_n outputs of the SIFB. The SIFB sends an IND+ event to indicate the presence of new data. In order to send response data, the IEC 61499 application must put them in the SD_1, … SD_m inputs of the SIFB and send a RSP+ event. The SIFB then transfer the results back to RD_1, … RD_m outputs of the SEND FB that pulses the NDR output to indicate that response data is present. The SEND FB has also a R input that is used to reset the procedure call before the response data is received. In

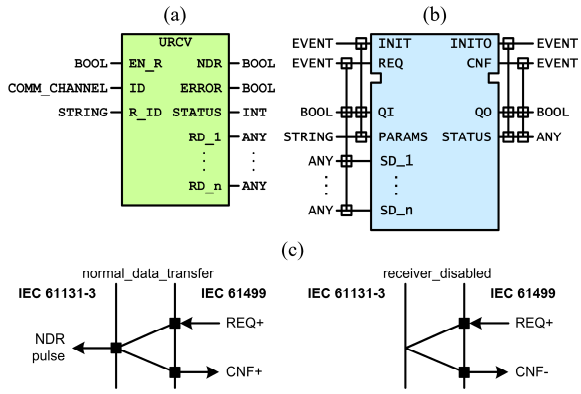


Figure 6. Data transfer PDE from IEC 61499 to IEC 61131-3: (a) IEC 61131-3 FB; (b) IEC 61499 SIFB; (c) time-sequence diagram.

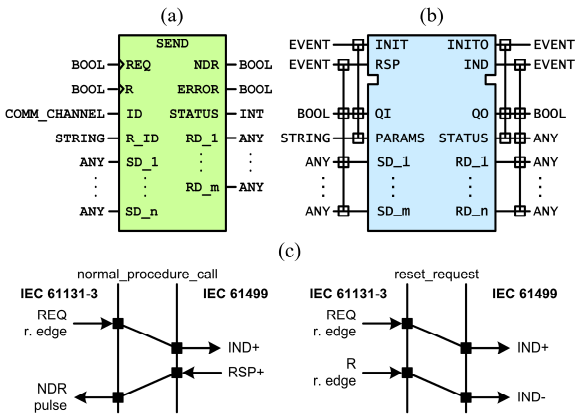


Figure 7. Procedure call PDE from IEC 61131-3 to IEC 61499: (a) IEC 61131-3 FB; (b) IEC 61499 SIFB; (c) time-sequence diagrams.

that case, the SIFB signals this behaviour to the application with an IND- event.

The Procedure Call PDE from IEC 61499 to IEC 61131-3 (Fig. 8) allows IEC 61499 applications to send input data to IEC 61131-3 programs and to receive response data from them. In order to do this, an instance of an IEC 61131-5 RCV FB (Fig. 8 (a)) must be present in the IEC 61131-3 program. That instance must be periodically invoked in order to check if new data are received from the IEC 61499 SIFB (Fig. 8 (b)). When data are received, the RCV FB instance waits until the IEC 61131-3 program computes response data and then it sends them back to the SIFB.

Fig. 8 (b) shows that the procedure call is initiated when a REQ+ event is received by the SIFB. The SD_1, … SD_m inputs of the SIFB are transferred to the RD_1, … RD_m outputs of the RCV FB (if it is enabled by EN_R) that pulses NDR to signal that data are received. Response data stored in the SD_1, … SD_n inputs of the RCV FB are transferred to the RD_1, … RD_n outputs

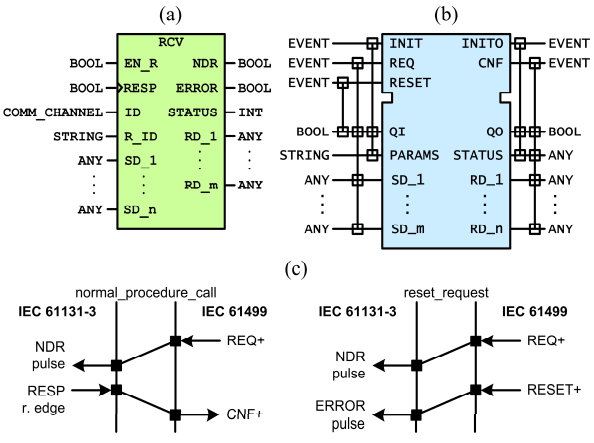


Figure 8. Procedure call PDE from IEC 61499 to IEC 61131-3: (a) IEC 61131-3 FB; (b) IEC 61499 SIFB; (c) time-sequence diagrams.

of the SIFB when a rising edge of the RESP input is detected. The SIFB sends a CNF+ event to signal that the response has arrived. The SIFB also has a RESET event input that allows to stop waiting for the results of the procedure call.

The behaviour of the IEC 61131-5 FBs is described in detail in the IEC 61131-5 standard [13]. The ID and R_ID inputs of IEC 61131-5 FBs are used to identify the particular IEC 61499 SIFB which should receive the data and the PDE to perform. ID must contain the channel descriptor of the PI, while R_ID must contain the identifier of the specific PDE.

5. Implementation details

A prototype of the proposed architecture has been realized and a simple system has been developed to test the functionalities of the architecture.

5.1. Used runtime environments and IDEs

The proposed solution has been implemented on a PC based environment with Microsoft Windows. The runtime environment used for execution of the IEC 61131-3 part of the system is ProConOS 4.0, developed by KW-Software [22] and the IDE (Integrated Development Environment) used to write code for ProConOS is MULTIPROG 4.8, also developed by KW-Software. Both ProConOS and MULTIPROG can be customized in order to add functionalities. ProConOS has been extended with custom-defined communication FBs written in C language to realize the PDEs.

The IEC 61499 part of the system is realized using the open source runtime environment FORTE, developed in the scope of the 4DIAC initiative [23]. The IEC 61499 application is developed using 4DIAC-IDE that can send IEC 61499 applications to FORTE. 4DIAC-IDE also allows to design interfaces for SIFBs that can be implemented in C++ language and included in FORTE. SIFBs realizing the PLC interfaces have been implemented in this way.

5.2. Implementation strategy

Data exchange between the two runtime environments is realized using inter-process communication techniques. In particular, a shared memory approach is used, where each PDE has its own data buffer to store input and output parameters separated from other PDEs. Synchronization between the SIFBs and the IEC 61131-5 FBs is realized using semaphores.

IEC 61131-5 FBs were developed and installed into ProConOS as C functions. Each invocation of the function block instance is executed as a function call and inputs, outputs and internal state are passed to this function as parameters. Each system call used in the IEC 61131-5 FBs is non-blocking, since the execution of IEC 61131-3 code is cyclic.

SIFBs for the IEC 61499 part are more complex and are designed in FORTE as C++ classes. The behaviour of a SIFB on the reception of events is defined by a particular function. However this is not sufficient to implement all the possible interactions, since SIFBs should also generate output events when a data exchange is initiated by IEC 61131-3. This is realized using an external event handler thread that waits for data from IEC 61131-3.

5.3. Test prototype

A test system has been designed to verify the feasibility of the architecture. The system is composed of three different personal computers running both ProConOS and FORTE. The test application realizes a distributed “and” between two boolean inputs (IN_1 and IN_2) sampled from two different computers (PLC_1 and PLC_2). The and function is calculated by a third personal computer (PLC_3) and the result is sent back to both PLC_1 and PLC_2 that update their outputs (OUT_1 and OUT_2).

Fig. 9 (a) shows the model of the system designed with IEC 61499 before the actual mapping on the devices (including initialization of FBs and insertion of communication FBs) is done. Each PLC interface abstracts a PLC and hides its implementation. The PI of PLC_1 (PI_1) includes only one PDE that realizes a procedure call from IEC 61131-3 to IEC 61499 with a boolean input parameter and a boolean output parameter, so it sends its input (IN_1) and waits for the response (OUT_1) before starting sending again. The PI of PLC_2 (PI_2), instead, uses an asynchronous approach, using two data transfer PDEs: one for sending IN_2 to IEC 61499 and one for receiving OUT_2. In this way, it is not necessary to wait for the response before sending the updated value of IN_2. The PI of PLC_3 (PI_3) includes a procedure call PDE from IEC 61499 to 61131-3 with two boolean inputs and one output that abstracts the and function. Since both data inputs of PI_3 are sampled only at the reception of the REQ event that is connected to PLC_1, it is necessary to store the last received value of IN_2 until it is sampled. This is done introducing a latch (E_D_FF) on PLC_3.

Fig 9 (b) shows the input and output values of PLC_1 and PLC_2 recorded using the MULTIPROG logic analyzer and demonstrates that the system correctly realizes the distributed and. Fig 9 (c) shows the IEC 61131-3 program running on PLC_1.

6. Conclusion and future work

In this paper, an architecture for the integration of IEC 61131-3 and IEC 61499 is presented. The architecture is based on parallel use of runtime environments of both standards and a method for defining interfaces for inter-standard data exchange is described in detail.

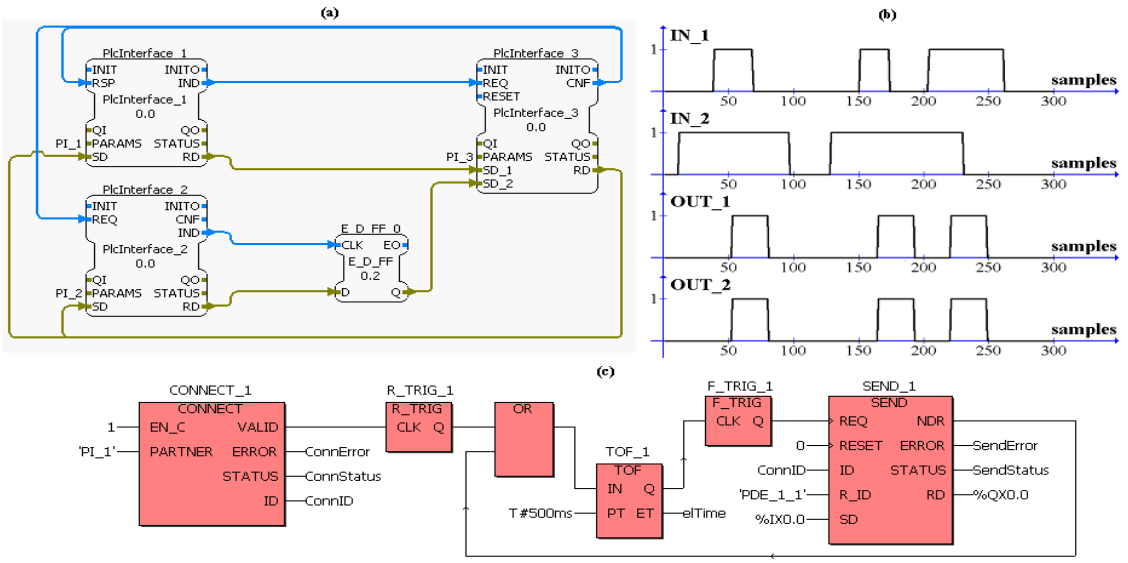


Figure 9. Test system: (a) IEC 61499 model of the system; (b) recordings of input/output of PLC_1 and PLC_2: (c) IEC 61131-3 program running on PLC_1.

An implementation of the architecture has been realized and a simple test case demonstrated the feasibility of the proposed solution.

In future works we plan to extend the test case to reference case studies derived from literature and industrial applications and to investigate code generation techniques for the implementation of PLC interfaces.

7. Acknowledgements

This project has been partially supported by ISAC s.r.l. and Regione Toscana under the "Bando Unico" framework. The authors would like to thank the 4DIAC team for their support and Daniele Nardi and Simone Genovese for their help in the test system.

References

[1] IEC 61131-3, "Programmable controllers – Part 3: Programming languages", International Electrotechnical Commission, 1993.

[2] M. Buchner, "Distributed Computer Control for Industrial Process Systems Characteristics, Attributes, and an Experimental Facility", *IEEE Control Systems Magazine*, Vol.2, pp. 8-15, 1982.

[3] IEC 61499-1, "Function blocks – Part 1: Architecture", International Electrotechnical Commission, 2005.

[4] V. Vyatkin, "IEC 61499 as Enabler of Distributed and Intelligent Automation State-of-the-Art Review", IEEE Transactions on Industrial Informatics, Vol.7, pp. 768-781, 2011.

[5] P. Tait, "A path to industrial adoption of distributed control technology", *3rd IEEE International Conference on Industrial Informatics*, pp. 86-91, 2005.

[6] M. Colla, A. Brusaferri and E. Carpanzano, "Applying the IEC-61499 Model to the Shoe Manufacturing Sector", *IEEE Conference on Emerging Technologies and Factory Automation*, pp. 1301-1308, 2006.

[7] K. Thramboulidis, S. Sierla, N. Papakonstantinou and K. Koskinen, "An IEC 61499 Based Approach for Distributed Batch Process Control", *5th IEEE International Conference on Industrial Informatics*, pp. 177-182, 2007.

[8] http://www.isagraf.com/.

[9] http://www.nxtcontrol.com/.

[10] K. Thramboulidis, "IEC 61499 in Factory Automation", *IEEE International Conference on Industrial Electronics, Technology and Automation (CISSE-IETA'05)*, Bridgeport, USA, 2005.

[11] K. H. Hall, R. J. Staron, and A. Zoitl, "Challenges to Industry Adoption of the IEC 61499 Standard on Event-based Function Blocks", *5th IEEE International Conference on Industrial Informatics*, pp. 823-828, 2007.

[12] A. Zoitl, T. Strasser, C. Sunder, and T. Baier, "Is IEC 61499 in Harmony with IEC 61131-3?", *IEEE Industrial Electronics Magazine*, Vol.3, pp. 49-55, 2009.

[13] IEC 61131-5, "Programmable controllers – Part 5: Communications", International Electrotechnical Commission, 2000.

[12] C. Gerber, H. M. Hanisch, and S. Ebbinghaus, "From IEC 61131 to IEC 61499 for distributed systems: a case study", *EURASIP J. Embedded Syst.*, pp. 1-8, 2008.

[13] W. Dai and V. Vyatkin, "Redesign distributed IEC 61131-3 PLC system in IEC 61499 function blocks", *IEEE Conference on Emerging Technologies and Factory Automation,* pp. 1-8, 2010.

[14] C. Sünder, M. Wenger, C. Hanni, I. Gosetti, H. Steininger, J. Fritsche, "Transformation of existing IEC 61131-3 automation projects into control logic according to IEC 61499", *IEEE International Conference on Emerging Technologies and Factory Automation)*, pp.369-376, 2008.

[15] G. Shaw, P. Roop, and Z. Salcic, "Reengineering of IEC 61131 into IEC 61499 Function Blocks", *8th IEEE International Conference on Industrial Informatics*, pp. 1148-1153, 2010.

[16] M. Wenger, A. Zoitl, C. Sunder, and H. Steininger, "Transformation of IEC 61131-3 to IEC 61499 based on a model driven development approach", *7th IEEE International Conference on Industrial Informatics*, pp. 715-720, 2009.

[17] Proposed annex H http://www.holobloc.com/stds/iec/sc65bwg7tf3/document/annex h_0.0.pdf

[18] V. Vyatkin and J. Chouinard, "On Comparisons of the ISaGRAF implementation of IEC 61499 with FBDK and other implementations", *6th IEEE International Conference on Industrial Informatics*, pp. 264-269, 2008.

[19] C. Sünder, A. Zoitl, J.H. Christensen, H. Steininger and J. Ritsche, "Considering IEC 61131-3 and IEC 61499 in the context of component frameworks", *6th IEEE International Conference on Industrial Informatics*, pp. 277-282, 2008.

[20] http://www.kw-software.com/

[21] http://www.fordiac.org/