Computers in Industry 72 (2015) 47–67

Contents lists available at ScienceDirect

Computers in Industry

journal homepage: www.elsevier.com/locate/compind

An architecture to integrate IEC 61131-3 systems in an IEC 61499 distributed solution

Stefano Campanelli [b], Pierfrancesco Foglia [a,*], Cosimo Antonio Prete [a]

[a] Dipartimento di Ingegneria dell'Informazione, Università di Pisa, Via Diotisalvi 2, 56126 Pisa, Italy
[b] ION Trading, Via Ceci 152, Pisa, Italy

ARTICLE INFO

Article history:
Received 30 May 2014
Received in revised form 14 April 2015
Accepted 22 April 2015
Available online 30 May 2015

Keywords:
PLC
SW reuse and integration
Distributed control systems
IEC 61131
IEC 61499
Discrete manufacturing
Manufacturing automation

ABSTRACT

The IEC 61499 standard has been developed to allow the modeling and design of distributed control systems, providing advanced concepts of software engineering (such as abstraction and encapsulation) to the world of control engineering. The introduction of this standard in already existing control environments poses challenges, since programs written using the widespread IEC 61131-3 programming standard cannot be directly executed in a fully IEC 61499 environment without reengineering effort. In order to solve this problem, this paper presents an architecture to integrate modules of the two standards, allowing the exploitation of the benefits of both. The proposed architecture is based on the coexistence of control software of the two standards. Modules written in one standard interact with some particular interfaces that encapsulate functionalities and information to be exchanged with the other standard. In particular, the architecture permits to utilize available run-times without modification, it allows the reuse of software modules, and it utilizes existing features of the standards. A methodology to integrate IEC 61131-3 modules in an IEC 61499 distributed solution based on such architecture is also developed, and it is described via a case study to prove feasibility and benefits. Experimental results demonstrate that the proposed solution does not add substantial load or delays to the system when compared to an IEC 61131-3 based solution. By acting on task period, it can achieve performances similar to an IEC 61499 solution.

1. Introduction

Programmable Logic Controllers (PLCs) are some of the most widespread devices in industrial automation, being used in industry for decades [30]. 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 among different devices. The new market demands [32] for flexibility and reconfigurability in manufacturing and process industries motivate the need of a transition from centralized to distributed control systems [2,4,31,41]. Distributed control is highly desirable in manufacturing industry, however, the design of a distributed control system is more challenging than a centralized one [4]. In particular, in an IEC 61131-3 environment, the aspects of distribution of control logic (such as the mapping of the code modules or variables to the devices and the communications) must be considered from the beginning of the design phase [40]. This complicates the design of the distributed control system and, moreover, it limits the ability to reconfigure the system, since changing the control system environment requires a redesign effort and the modification of existing software modules [40].

In order to facilitate the design of distributed control systems, IEC proposed the IEC 61499 standard [3]. The standard introduces in the development of distributed control application advanced concept derived from distributed systems and the software engineering discipline, such as encapsulation, independence of the control logic from the communication infrastructure, and development of software component independently from their mapping to hardware. The standard defines an open architecture, based on an event driven execution approach, that permits to model and design control applications organized in software components distributed across networked devices [4]. Practical case studies [5,6,7] have proved the benefits of the standard, and

* Corresponding author. Tel.: 0039 0502217530.
E-mail addresses: ste.campanelli@gmail.com (S. Campanelli), foglia@iet.unipi.it (P. Foglia), prete@iet.unipi.it (C.A. Prete).

http://dx.doi.org/10.1016/j.compind.2015.04.002
0166-3615/

there are some commercial tools supporting it already on the market (ISaGRAF [8] and nxtControl [9]).

Since IEC 61131-3 has been in use for years [30], there are both a large number of control systems in operation and already developed and tested software libraries that are based on this standard. Unfortunately, it is not possible to directly execute IEC 61131-3 applications in an IEC 61499 based runtime environment [4,11], so the adoption of the IEC 61499 would result in the waste of such existing code, know-how and competences, as also noted for the batch process industry and the IEC 61512 standard [33]. Besides, the adoption of IEC 61499 in an existing system would result at least in the modification of the software implementation of the control algorithm, with consequent need of new design and testing phases and interventions on the production line with the eventual stop of operation. The switching cost to the new standard is then perceived to be very high, and it could be lowered if some of the IEC 61131-3 investments are retained [33]. For these reasons, companies that have been already slow to adopt languages other than IEC 61131-3 [30], are even less inclined to convert existing systems to IEC 61499 [31], nevertheless, many of these systems still may take advantage from distribution of control logic to increase interoperability, reduce human work, improve control decisions, reliability, scalability, reconfiguration and deployment [2,4,31,41]. One solution to minimize the cost, retain the investment and exploit distribution consists in developing some components in the new standard, while keeping the other software modules already written in the old.

For instance, starting from an existing IEC 61131-3 system, there is the need to redesign new components to allow a more flexible distributed control when the application is running on more devices. In such a situation, it is convenient to adopt, for the new components, the IEC 61499 standard, while reusing, to retain the investment, the available IEC 61131-3 code. In a different use case, starting from an IEC 61131-3 system, the introduction of new hardware components determine the introduction of new software modules devoted to their management. Such modules can be written again in the new standard, to exploits the benefit of it. As a general observation on the advantage of coexistence of both the standards, we can say that the IEC 61131-3 standards permits to see software modules as hardware-like modules (ladder diagrams or FB diagrams), while the IEC 61499 standard gives the system a more software oriented vision, with components, connections and events. The IEC 61499 vision is useful for system designers, who are usually experienced software/control engineers and have to deal with the distributed aspects of control. Nevertheless, the hardware-like vision is useful for plant staff, which, although inexperienced in advanced control engineering techniques, need to keep the plant operational [11,30].

Such examples motivate the needs of architectures that permits the coexistence of code modules written in the two standards [12]. Such coexistence is not simple to realize by system designers, as IEC 61131-3 modules can't be directly executed on IEC 61499 runtime [4,11].

In this paper, we deal with such problem, and we propose an architecture, namely LE-INT (LowEffort-INTegration architecture) to realize the coexistence of code modules written in the two standards without requiring, for system designers, effort and knowledge higher to the ones required to write software modules in the two standards. In particular, the proposed solution does not require modifications to the run-time support of the two standards, as well as modification to the development tools, as it utilizes defined features of the two standards.

In the proposed solution, there are physical devices able to execute IEC 61131-3 and IEC 61499 code modules. Such modules can be on the same device or on different physical devices; they can exchange information through appropriate mechanisms that encapsulates the interaction and the implementation details of the communications. As a consequence, IEC 61131-3 developers and staff people do not need to know the details of IEC 61499 code, that is abstracted as IEC 61131-5 function blocks, with no loss of acquired competence and development effort. From the IEC 61499 perspective, the IEC 61131-3 control logic is abstracted as one or more Service Interaction Function Blocks (SIFB), that can be utilized as standard IEC 61499 modules, adhering to the concept of encapsulation of the standard.

Based on LE-INT, we define a methodology for the integration of IEC 61131-3 code modules in IEC 61499 logic. In this way, as we will show with a case study, existing IEC 61131-3 code can be kept and easily integrated in an IEC 61499 distributed solution.

Main fields of application of LE-INT are the manufacturing and automation processes (such as, but not only, discrete part production lines [37]) organized in a modular way and as sequences of processing steps. Such systems may have a not sequential information flow and may include branching paths. Examples can be found in [37,38,39]. LE-INT is effective in these systems, to realize the following use scenarios (use cases):

(1) transition to distributed mixed systems using both standards from existing IEC 61131-3-based systems;
(2) insertion of one or more existing IEC 61131-3 system in an existing distributed IEC 61499 system;

In the first use-case, LE-INT can be used to transform a centralized or several independent IEC 61131-3 systems in a distributed system based on both IEC 61131-3 and IEC 61499 standards. In this scenario, LE-INT permits to use the distribution features of IEC 61499, while reusing existing IEC 61131-3 code. A limit of our approach consists in the need of performing a decomposition of the existing system (in case of a centralized system) that is not straightforward and requires effort. However, when the manufacturing process is organized in distinct steps [38] or the system is composed of different physical subsystems [39], usually the centralized system is structured in a modular way where different code modules manage different parts of the physical system, communicating via global variables [39]. In these cases, the effort of decomposing the system is greatly reduced, since code is already decoupled.

The second use-case refers to the insertion of an existing IEC 61131-3 module in a modular distributed production-line programmed in IEC 61499, as shown in [39]. In this case, LE-INT allows to see, from the IEC 61499 point of view, the IEC 61131-3 system as a software component (a SIFB) that can become part of the distributed application. The disadvantage of this approach consists mainly in the delay added by the mixing of cyclic and event-driven execution models, delay that we will evaluate in the paper, with reference to a case study.

Results of the evaluation indicate that LE-INT performs similarly to a pure IEC 61131-3 solutions and, by acting on task period, it can achieve performance similar to a pure 61499 solution, while the overhead in terms of CPU and memory usage, introduced by the contemporary execution of two runtimes, in negligible. As a summary, the advantages introduced by LE-INT (derived from the opportunity of reuse, distribution and expansion) are not overwhelmed by performance drawbacks.

The rest of the paper is structured as follows: Section 2 presents related works, basic concepts of IEC 61131-3 and IEC 61499 are discussed in Section 3, the proposed architecture is described in Section 4, in Section 5 a methodology of integration is presented via a case study, while Section 6 discusses implementation and performance evaluation. Section 7 concludes the paper.

2. Related work

A review of the research activity around the IEC 61499 standard is presented by Vyatkin in [4], while a preliminary work on its diffusion is presented by Thramboulidis in [10]. Works about the coexistence of the standards and migration from IEC 61131-3 to IEC 61499 are particularly relevant for the problem addressed in this paper.

Zoitl et al. [12] discussed 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, the existing work dealing on the first approach can be summarized in a proposed annex [19], whose contents have been now included in the second edition of IEC 61499 standard [3] in the informative Annex D.6. Such an annex proposes a set of IEC 61499 function blocks for interoperability with IEC 61131-5 compliant devices.

Regarding the others two approaches, ISaGRAF [8] is a commercial solution that utilizes the second approach, implementing the IEC 61499 concepts in an IEC 61131-3 environment. This implementation has some differences from the other implementations of the standard (analyzed by Vyatkin and Chouinard in [20]). Sünder et al. [21] discussed the possibility of a component based implementation of the third approach. In addition, nxtSTUDIO is an IEC 61499 commercial tool developed by nxtControl [9] that recently introduced the opportunity to code program parts using IEC 61131-3 following the third approach.

In this work, we choose the first approach. In a preliminary paper [26], we presented the initial ideas of an architecture that follows such an approach, but lacking case studies and methodologies to prove the benefits of it. To point out the differences between our work and the annex D.6 of IEC 61499 [3], the function blocks introduced by the annex assume that interactions are performed following a client/server model where IEC 61131-3 PLCs can act as servers that respond to requests by IEC 61499 clients [19]. The SIFBs introduced to perform the communication are READ, UREAD, WRITE and TASK, based on IEC 61131-5 [13] primitives with the same names. Referring to the terminology introduced in IEC 61131-5, these function blocks allow to perform interactions described as "polled data acquisition" and "parametric control" [13]. As explained in Section 4.2, our proposed approach, instead, is based on the USEND, URCV, SEND and RCV IEC 61131-5 function blocks that allow to perform "programmed data acquisition" and "interlocked control" interactions. From this point of view, our approach is complementary to the one proposed in the annex D.6 ([3,19]) and, together, they allow to realize all the communication models included in the IEC 61131-5 standard. Besides, in our approach, both the IEC 61131-3 PLC and the IEC 61499 devices can act as client/server, and this is more adherent to the IEC 61131-5 and IEC 61499 specifications.

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 61499 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 redesign the entire system. Both papers also give some general guidelines to simplify future migration processes. Similarly, Wenger et al. present in [16] and [25] some transformation rules to manually migrate a system. Even, though recommendations, guidelines and transformation rules can facilitate the migration process, a re-engineering process is still required, so 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]. Dai et al. [45] propose a new approach of migration from IEC 61131-3 to IEC 61499 utilizing semantic web technologies. However, many of these approaches are incomplete and they are still not sufficiently compatible to convert a system without further human work. Another limit of these approaches is that if the IEC 61131-3 systems make use of vendor-specific code modules (that usually are closed-source and allow to access to complex features of hardware modules furnished by the manufacturers), such code modules should be ported in the IEC 61499 runtime, and this operation may require significant effort. Moreover, in some cases the code automatically generated has poor structure and it is not very readable, so it would be difficult to maintain. Above all, the migration approach may entail the loss of the IEC 61131-3 code, and then it may not solve the disadvantage of adopting standards other than IEC 61131-3 [30].

Many recent works are focused on techniques to improve the programmability and reuse of modern distributed control systems. Yang and Vyatkin propose transformation of Simulink models to IEC 61499 FB to help verification and validation of distributed control systems [31], while Bengtsson et al. introduce methodologies to improve the reuse of PLC code [30]. Hästbacka et al. and Thramboulidis et al. provide means for developing control applications using domain-specific modeling concepts to increase productivity and enhance solutions reuse [32,34]. Bonfé et al. present object oriented modeling approaches and design patterns supporting the implementation of industrial control systems [35], while a software architecture based on mobile agents for distributed process control applications has been proposed by Di Stefano et al. [29]. All of such works are orthogonal to our work, in that they provide mechanisms that can integrate our solution.

3. Basic concepts

3.1. IEC 61131-3 basics

The IEC 61131-3 [1] standard defines a set of programming languages (three graphical and two text-based) as well as some common elements to all programming languages for the development of applications for programmable controllers. Such applications can be concentrated or distributed on more devices.

The common elements include definition of data types, variables, Program Organization Units (POUs) and a software model of the PLC, used to organize the execution of the program code into a set of tasks. 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 a sample IEC 61131-3 FB is shown in Fig. 1a.

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 extend the concept of FB defined in IEC 61131-3 adding event inputs and outputs in addition to data inputs/outputs. In Fig. 1b, 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

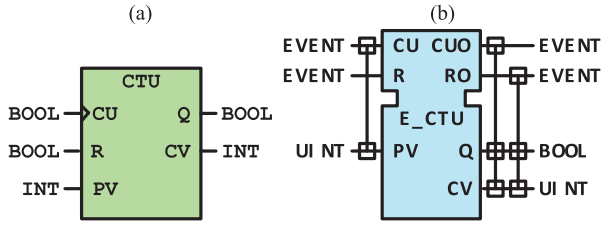


Fig. 1. A simple counter function block realized in both standards. The IEC 61131-3 FB (a) counts rising edges of CU data input. The IEC 61499 FB (b) counts the occurrences of CU event.

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 two or more FBs.
- Service Interface Function Block: it allows to utilize services provided by the device whose implementation is hidden and its interface is described using time-sequence diagrams. It is often used to implement low-level functionalities such as communication or user interface.

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

3.3. Execution models of the two standards

In IEC 61131-3, the developer defines the tasks of the system and associates the program or function block instances to these tasks. When a task is activated for execution, the associated function blocks are executed in a designed order. The tasks can be of three kinds:

- cyclic tasks: they are executed cyclically, that means that the code modules associated to the task are executed and when the task completes its execution it starts again;
- periodic tasks: they are executed at regular time intervals; the developer choose the period of task activation and the associated modules are executed at each activation;
- event tasks: they are executed at the occurrence of particular events such as the change of value of a particular boolean condition, in that case, when the event occurs the code modules associated to the task are executed once.

The most common task types are periodic and cyclic, because they are based on the typical scan-based execution approach of the PLCs, where the following steps are executed cyclically: reading of the PLC inputs, execution of control logic, updating of the PLC outputs. For this reason, most function blocks (such as IEC 61131-5 communication function blocks) are designed to work with cyclic or periodic execution.

The IEC 61499 execution model uses a pure event-based approach where a function block is executed when it receives an input event and, after its execution, it may generate one or more output events connected to other function blocks. In this case, the execution order is not fixed like in IEC 61131-3 but depends on the sequence of events. For example, in IEC 61499, if a function block never receives an event, it will never be executed, while in IEC 61131-3 the function block is executed at each task activation whether its computation is needed or not.

These execution models are profoundly different and each method has its advantage over the other one: the event-driven approach of IEC 61499 allows better performance (only the function block that actually have work to do are executed), but the fixed execution of IEC 61131-3 is more predictable, fundamental aspect in the developing of real-time applications [4].

3.4. Support to communication

IEC 61131-3 tasks running on the same device can communicate via shared memory (shared variables). The IEC 61131-3 software model defines access paths that allow to access PLC variables from other PLC via communication services while the IEC 61131-5 standard [13], another part of the IEC 61131 standard, introduces the communication services of PLCs. It defines a set of communication paradigms and communication function blocks that can be utilized in IEC 61131-3 programs to exchange data among different PLCs connected through a network. In the following (Fig. 2), two of the communication paradigms described in IEC 61131-5 are presented, since their concepts are used in the proposed architecture: the programmed data acquisition and the interlocked control. Programmed data acquisition allows to transfer data from a PLC to another PLC and the transfer is initiated by the sender PLC. Such operation is useful, for instance, to signal to another PLC the state change or new data acquired from a sensor. Interlocked control is used to realize a sort of remote procedure call between different PLCs, where a PLC requests a computation to another PLC, sending some parameters and waiting for results. Such operation is useful, for instance, when a PLC needs to move in an assigned position (specified as input parameter) an arm controlled by another PLC, and waits for the successful/unsuccessful completion of the operation. Fig. 2a shows the IEC 61131-5 communication function blocks that realize the

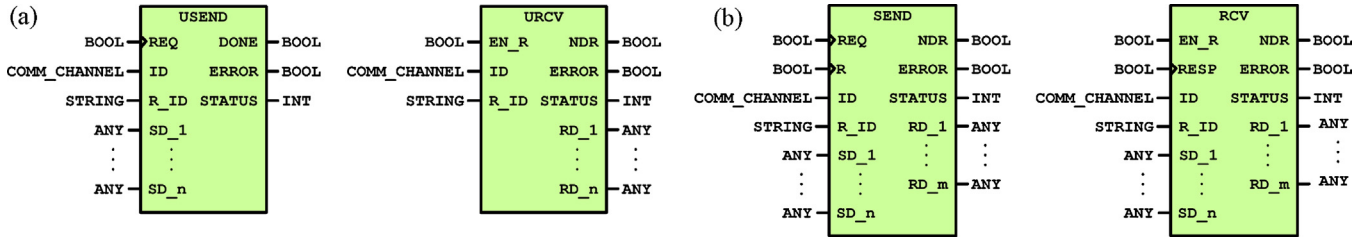


Fig. 2. IEC 61131-5 communication function blocks for programmed data acquisition (a) and interlocked control (b).

programmed data acquisition paradigm to send data from a sender PLC (that has an instance of the USEND FB) to a receiver PLC (that has an instance of the URCV). The communication is initiated by the sender PLC when a rising edge is detected on the REQ input of the USEND instance, then data (SD_1, ... SD_n) are transferred to the receiver PLC, then the NDR output of the URCV instance pulses to indicate the data reception (RD_1, ... RD_n). Fig. 2b shows the communication function blocks that realize the interlocked control communication paradigm. This type of communication is used when the sender PLC (that has an instance of the SEND) requests the receiver PLC (that has an instance of the RCV) for executing an operation and wants to know the results. The communication is initiated by the sender when a rising edge is detected on the REQ input of the SEND instance, then the parameters of the operation (SD_1, ... SD_n) are sent to the receiver PLC. When data arrives the NDR output of the RCV instance pulses and the parameters are available to the program on the RD_1, ... RD_n outputs of the RCV instance. The program on the receiver PLC then calculates the results of the requested operation and sends them back to the receiver by putting them on the SD_1, ... SD_m inputs of the RCV instance and raising the RESP input. When result data arrives to the sender PLC, the NDR output of the SEND instance pulses and the results are available in the RD_1, ... RD_m outputs.

In the IEC 61499 standard, two communication paradigms are defined: the publish/subscribe and the client/server. The first paradigm is used for unidirectional communication of data between devices, while the second one is used for bidirectional communications. As an example, the function blocks used to realize the publish/subscribe model are shown in Fig. 3. The transfer is initiated when the PUBLISHER FB receives a REQ event: the data on SD_1, ... SD_n data inputs are sent through the network and, when done, a CNF output event is issued to inform the application. When the SUBSCRIBE FB receives the data, it informs the application with an IND event that data are available on the RD_1, ... RD_n outputs of the function block. When data are read, the application notifies the SUBSCRIBE FB with a RSP event.

Differently from IEC 61131-3, the IEC 61499 standard includes advanced concepts of distributed systems and software engineering. In particular, it allows the design of distributed control application without considering the implementation aspects of communication, and it decouples the design phase of software from its mapping phase, a basic concept to achieve software reuse. In the IEC 61499 standard, in fact, the control application is seen as a network of function blocks that exchanges data and events, and it is designed without considering the hardware infrastructure and the distribution aspects. Only after the design phase, the function blocks of the application are mapped to the devices. In this phase, communication function blocks are inserted between connected function blocks that are mapped to different devices [42]. Using this approach, the designer can build in an easier way the application, as (s)he does not have to consider the distribution aspects [46], while reconfigurability is simplified with respect to the IEC 61131-3, since the mapping of the function blocks to the devices can be easily changed without redesign effort.

4. LE-INT architecture

We consider (Fig. 4) a control system composed of different devices that interact to realize the control law. Our proposal provides execution capabilities for IEC 61131-3 tasks and IEC 61499 applications that are executed in full compliance with the standard they belong to (Fig. 4a).

LE-INT code modules of different standards can communicate only via specific FBs (Fig. 4b), and there are no constraints on where the couples of FBs are allocated. The idea consists in providing two mechanisms: the first one is a communication channel from a module written in one standard to a module written in the other standard; the second one, consists in requesting a service offered by modules of the other standard, specifying some input parameters, and waiting for the results.

As for the implementation, the code modules that belong to different standard can communicate via the programmed data acquisition and interlocked control paradigms defined in the IEC 61131-5 standard [13]. In particular, our architecture is based on a set of FBs (on the IEC 61131-3 side) and SIFBs (on the IEC 61499 side) available to the programmer to realize communications. FBs and SIFBs for a specific interaction are automatically generated. In particular, we utilized the USEND, URCV, SEND and RCV IEC 61131-5 standard FBs.

Fig. 5a shows the basic concept of a programmed data acquisition operation initiated by a IEC 61131-3 task that sends, via a USEND FB, data to the corresponding IEC 61499 SIFB. Such operation is useful, for instance, to signal to the high level distributed IEC 61499 application the state change of a sensor. When the IEC 61131-3 program needs to send data, it puts the data on the DATA_IN input and requests the operation to the USEND FB via the REQ input. The architecture then moves the data, transparently to the programmer, to DATA_OUT output of the IEC 61499 SIFB, and an IND event is generated to signal the application.

Fig. 5b shows the basic concept of a programmed data acquisition operation initiated by a IEC 61499 application, that sends data to the corresponding IEC 61131-3 URCV FB. When the IEC 61499 application needs to send data, it puts the data on the DATA_IN input and requests the operation to the SIFB via the REQ input. The architecture then moves the data, to DATA_OUT output of the IEC 61131 FB, and the NDR output is set to indicate that a new data has been received.

Fig. 6 shows the behavior in case of interlocked control, i.e. when the IEC 61499 application needs to send data and wait for an answer from the IEC 61131-3 program. This is useful when the IEC 61499 application needs a service that is available on the 61131-3 part of the application, and it can send a request with some input parameters and expect a response with some outputs, in a way

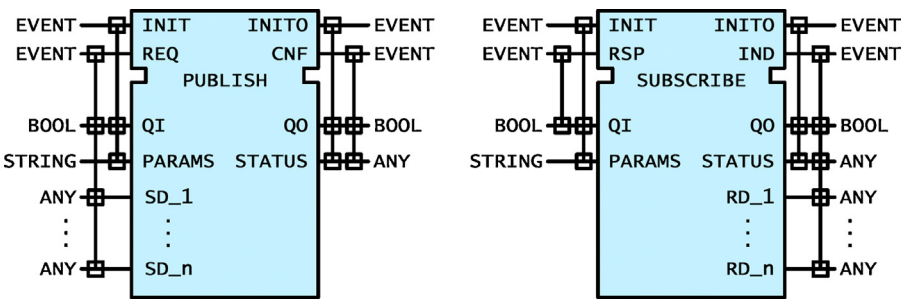


Fig. 3. IEC 61499 function blocks that realize the publish/subscribe communication model.

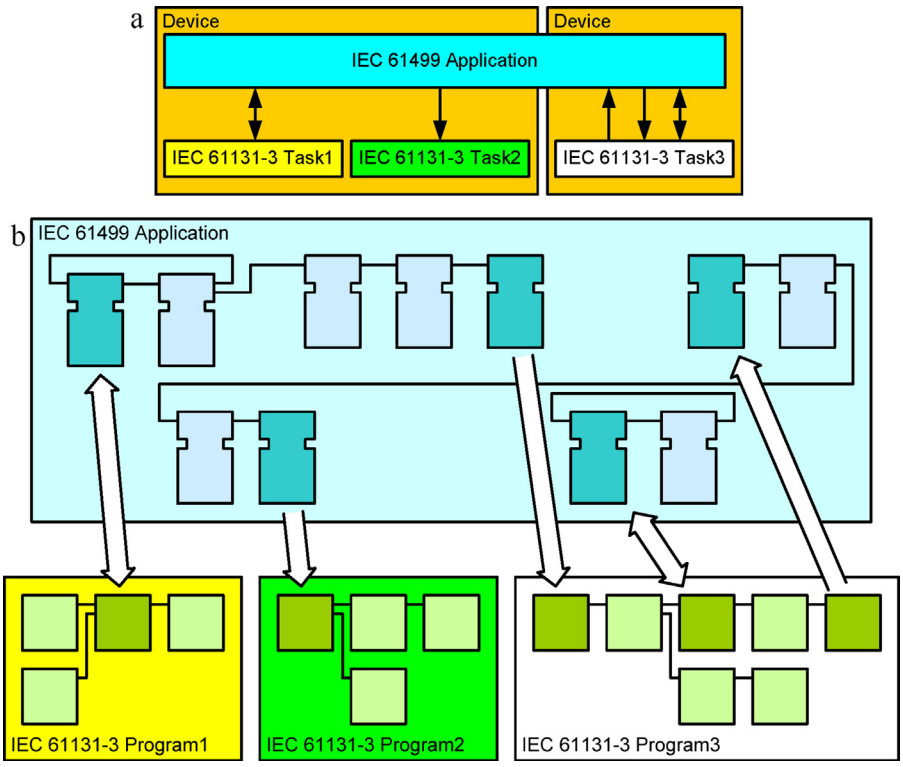


Fig. 4. (a) Reference System. (b) An example of control application with communications between modules written in two different standards. Code modules of different standards communicate only via specific FBs (the darker blue/green in figure). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

similar to a remote procedure call. When the application requests the services (Fig. 6a), it put the parameters on the PAR_IN lines and send a REQ event to the SIFB. The architecture then moves the data, transparently to the programmer, to the PAR_OUT output of the IEC 61131 RCV FB, which, when executed, set NDR to signal the program that new data are ready. The IEC 61131-3 program then executes its logic, and signals (Fig. 6b) the completion of the service execution using the RESP input of the RCV FB. Response data present in the DATA_IN inputs are then sent to the DATA_OUT lines of the SIFB, which signals the conclusion of the service issuing a CNF output event to the application. The dual case of interlocked control initiated by a IEC 61131-3 task works in a similar way, by utilizing the SEND FB and the specific SIFB (Fig. 7).

4.1. PLC data exchanges

As a summary, the proposed communication architecture is based on sets of pairs of IEC 61131-5 FB and 61499 SIFB, where input/output from FB are connected to output/input of SIFB, and a communication paradigm is defined on the pair. Such pairs are put in the IEC 61131-3 program and in the IEC 61499 application respectively, to implement the communication, as shown if Fig. 4b. We define each of these pairs as PLC Data Exchanges or PDEs.

Figs. 8–11 summarize the 4 types of PDEs furnished by our architecture, giving information about the function blocks of both standards needed to perform the data exchange and their interactions. A detailed description of the Programmed Data Acquisition PDE from IEC 61131-3 to IEC 61499 (Fig. 8) is provided below. For the other PDEs, only some general information is provided while the rest can be deduced in a similar way from Figs. 9–11 and Figs. 5 and 6.

The Programmed Data Acquisition PDE from IEC 61131-3 to IEC 61499 (Figs. 5a and 8) allows IEC 61131-3 programs to send data to IEC 61499 applications. An instance of IEC 61131-5 USEND

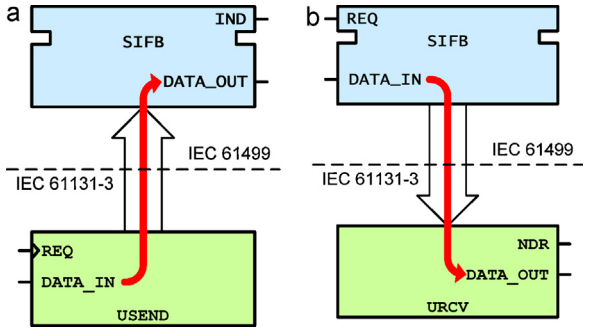


Fig. 5. (a) Programmed data acquisition initiated by a IEC 61131-3 task. (b) Programmed data acquisition initiated by an IEC 61499 application.

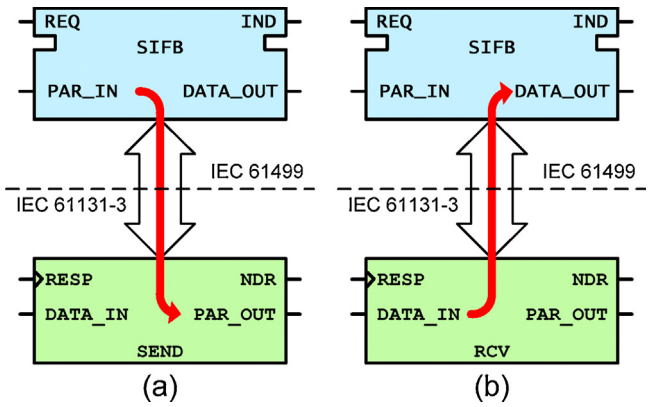


Fig. 6. Interlocked control initiated by an IEC 61499 application.

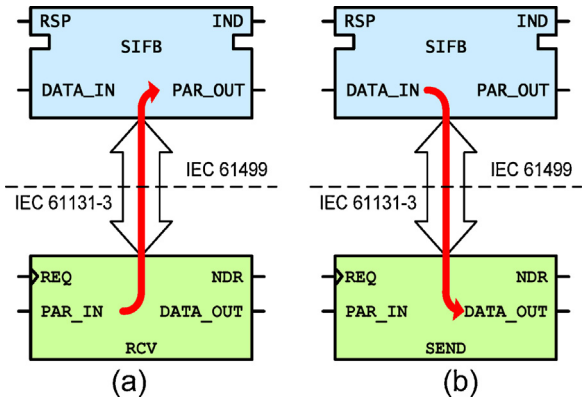


Fig. 7. Interlocked control initiated by a IEC 61131-3 task.

function block, shown in Fig. 8a, is used to communicate data to an instance of the associated SIFB shown in Fig. 8b. The USEND FB has a REQ input, that is used to request the data transfer; a DONE output that signals the completion of the transfer operation; R_ID and ID inputs that identify the PDE (they are automatically managed by the software) and SD_1, ... SD_n inputs that hold the data to be transferred. Such inputs correspond to the DATA_IN input of Fig. 5a.

The SIFB has an IND event output for signaling data reception; the PARAMS inputs that are the PDE identifier, and RD_1, ... RD_n data outputs that hold the received data. Such outputs correspond to the DATA_OUT output of Fig. 5a. According to the standard, the SIFB has also INIT/INITO events for initialization and the input and output qualifiers (respectively QI and QO) are used to giving a positive or negative characterization to an event, according to their boolean state.

Fig. 8c shows the time-sequence diagram of the Programmed Data Acquisition PDE. When a rising edge is detected on the REQ input of the USEND FB, the data are transferred to the SIFB. The SIFB then triggers 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 Programmed Data Acquisition PDE from IEC 61499 to IEC 61131-3 (Figs. 5b and 9) allows IEC 61499 applications to communicate data to IEC 61131-3 programs. From the IEC 61131-3 side, an instance of the URCV FB is used. The URCV FB has an EN_R input that allows to enable/disable reception of data and a NDR output that is used to signal the reception of new data. The IEC 61499 SIFB has a REQ event that is used to request data transfer and a CNF event that is issued when the data transfer has completed.

The Interlocked Control PDE from IEC 61131-3 to IEC 61499 is used when an IEC 61131-3 program needs a service that is available on the IEC 61499 part of the application (Figs. 7 and 10). The IEC 61131-5 SEND FB has a REQ input for requesting the service and a NDR output that indicates that response data is arrived. Also, a R input may be used to cancel the service request. The IEC 61499 has a IND event output that indicates when a procedure call is requested and the application must compute response data and issue an RSP event to conclude the handshake. A similar mechanism is used for Interlocked Control PDEs from IEC 61499 to IEC 61131-3, shown in Fig. 11.

Further details about initialization procedures can be found in the authors' previous paper [26].

4.2. Interaction function blocks

In order to simplify the design of the distributed application, we can use the proposed mechanisms to build more complex communications mechanisms, in such a way that the communications logically related to specific IEC 61131-3 components (for instance a program or the set of programs running on a resource) can be seen in the IEC 61499 application as a single SIFB (this is a typical solution adopted in the IEC 61499 standards to expose to an applications the set of services offered by a single device). An example is given in Fig. 12, where the three IEC 61131-3 program3 communications are seen as a single SIFB in the IEC 61499 application.

We will refer to this SIFB as Interaction Function Block (IFB), where each IFB involves a set of PDEs.

Fig. 13 shows an example of an Interaction Function Block constituted by the three PDEs (with reference to the Interaction Function Block presented in Fig. 12) (1) PDE A—a Programmed Data Acquisition PDE sending data from IEC 61499 to IEC 61131-3; (2) PDE B—an Interlocked Control PDE realizing a procedure call from IEC 61131-3 to IEC 61499; (3) PDE C—a Programmed Data Acquisition PDE sending data from IEC 61131-3 to IEC 61499.

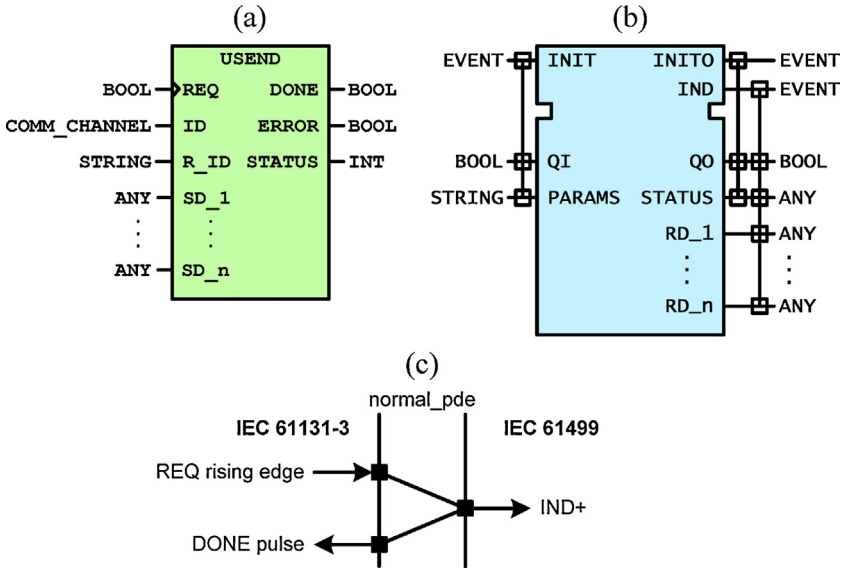


Fig. 8. Programmed Data Acquisition PDE from IEC 61131-3 to IEC 61499: (a) IEC 61131-3 FB; (b) IEC 61499 SIFB; (c) time-sequence diagram.

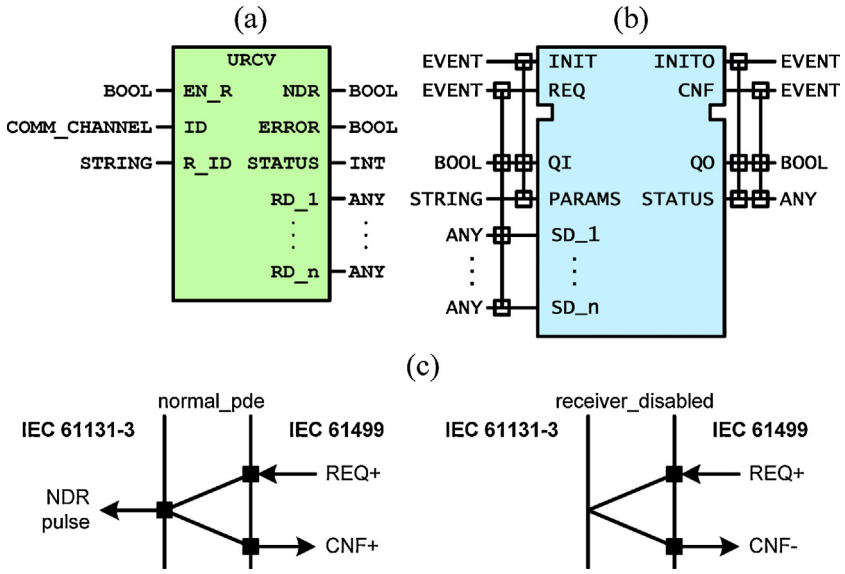


Fig. 9. Programmed Data Acquisition PDE from IEC 61499 to IEC 61131-3: (a) IEC 61131-3 FB; (b) IEC 61499 SIFB; (c) time-sequence diagrams.

Each event input/output of the IFB (Fig. 13a) is associated to a single PDE. In the figure this association is indicated appending A, B or C according to the particular PDE at the end of the event identifier. Connections between event and data input/outputs indicates the parameters to be exchanged: for example, the association between REQ_A event input and P1 data input indicates that P1 holds the value that the IFB must send to the IEC 61131-3 FB needed to implement PDE A.

Fig. 13b shows the interface of the three associated IEC 61131-5 FBs. The IEC 61131-5 communication function blocks must be associated to the particular IFB and PDEs in order to properly communicate with the IEC 61499 application. This association is done using the identification inputs (ID and R_ID) of the IEC 61131-5 function blocks. In particular the ID input is used to identify the associated IFB, while R_ID is used to identify the particular PDE.

5. Methodology and case study

Basing on the proposed architecture and the PDEs described in the previous section, it is possible to derive a methodology for the integration and/or the coexistence of IEC 61131-3 and IEC 61499 logic. In this section, such methodology is described, with reference to a case study.

5.1. Description of the test case

A production chain is considered as test bed. It consists of a series of mechatronic units that work on a workpiece in successive steps. Two particular units, shown in Fig. 14, and their interactions are considered in this case study. The unit on the left side is the feeder unit. It has a magazine of workpieces and a pusher to push workpieces out of the magazine, making them available for other

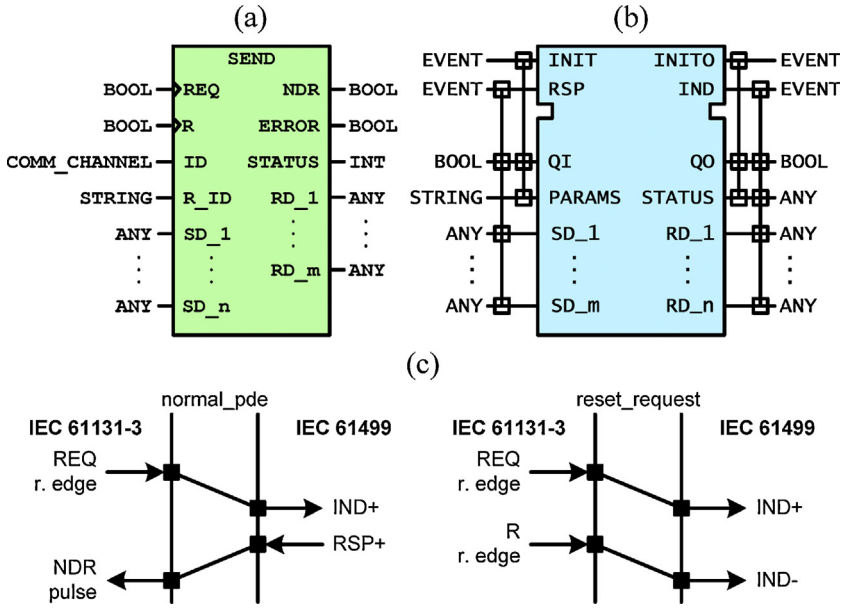


Fig. 10. Interlocked Control PDE from IEC 61131-3 to IEC 61499: (a) IEC 61131-3 FB; (b) IEC 61499 SIFB; (c) time-sequence diagrams.

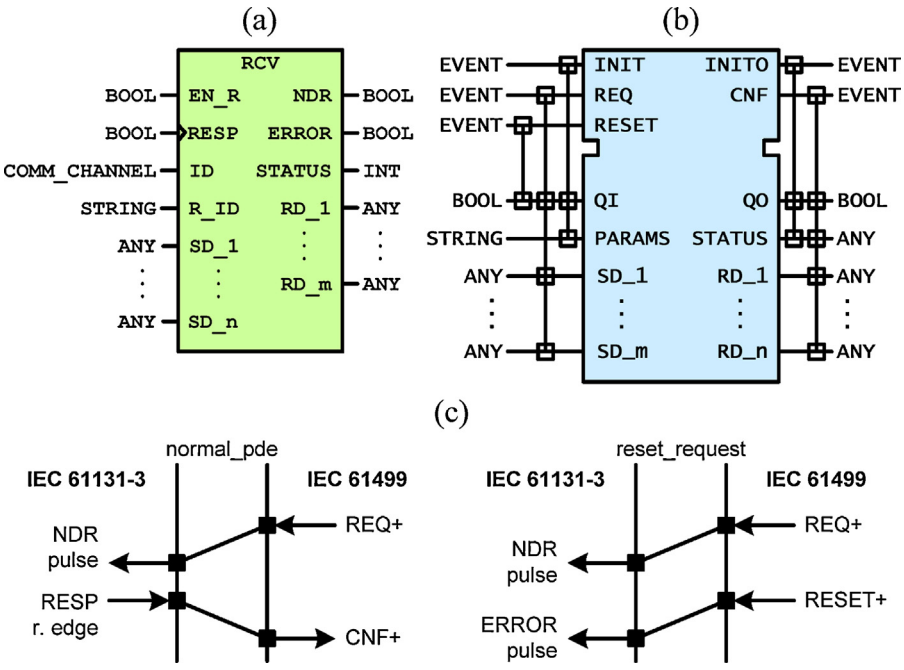


Fig. 11. Interlocked Control PDE from IEC 61499 to IEC 61131-3: (a) IEC 61131-3 FB; (b) IEC 61499 SIFB; (c) time-sequence diagrams.

units. On the right side, there is a transfer unit, that is a manipulator that moves workpieces from a left position to a right position. In this case, the transfer unit is used to take workpieces provided by the feeder and to transfer them to the next position, where they are supposed to be processed by other automated machines. In the initial configuration, the units are controlled using two independent IEC 61131-3 systems and each unit has no information regarding the status of the other one. A human operator is needed to give orders to the units for each workpiece to prevent clashes between mechanical parts. In order to automate this task, a proper control application must be designed. This control application needs information from both the feeder and the transfer units, so it is particularly suitable to be designed as a distributed control application using IEC 61499.

This test case is largely inspired by the one presented in [24] (derived from [27]) but there are some substantial differences. In [24], both units (feeder and transfer) are controlled by a single centralized system and the authors show how it is possible to redesign the control logic using IEC 61499 in order to allow distribution. In this paper, each unit is initially controlled by its own control system with independent control logic and we show how it is possible to add IEC 61499 distributed control logic by reusing the components written in the IEC 61131-3 standard.

5.2. Initial configuration

In the initial configuration (Fig. 14), the feeder unit and the transfer unit are independent. The control panel of the feeder units has three buttons and three LEDs: the START and STOP buttons allow to start and stop the feeder unit; the PUSH button pushes a workpiece out of the magazine; the ON LED indicates if the feeder unit is enabled, the PUSHED LED signals that a workpiece has been pushed out of the magazine and the EMPTY LED indicates that there are no more workpieces in the magazine.

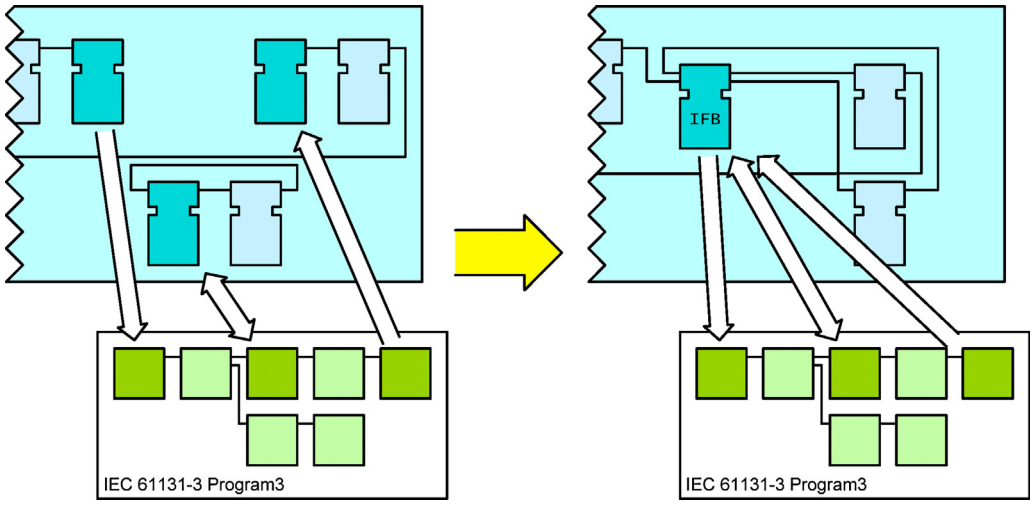


Fig. 12. The concept of Interaction Function Block (IFB).

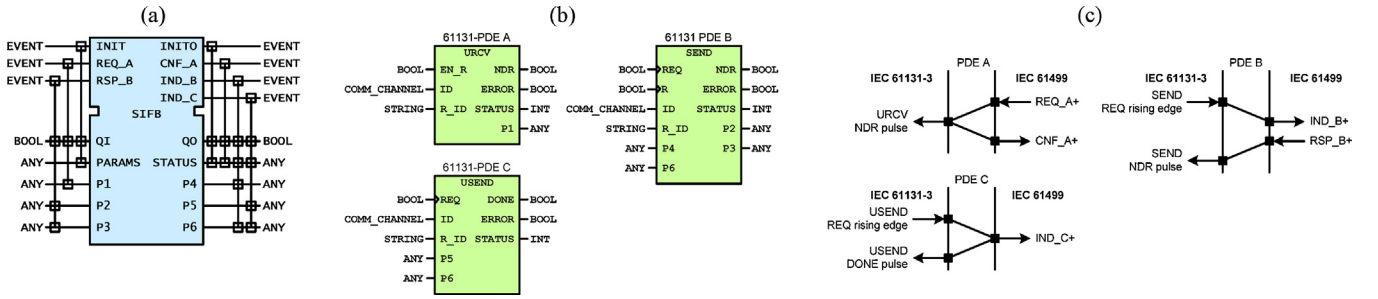


Fig. 13. Example of an Interaction Function Block composed of three PDEs. (a) Interaction Function Block Interface; (b) corresponding IEC 61131-3 FB interfaces; (c) time-sequence diagrams.

The transfer unit control panel has four buttons and three LEDs: the START and STOP buttons allow to start and stop the transfer unit; the TRANSF button moves the manipulator arm to the left position, takes the workpiece from the left to the right position, leaves it in the right position and moves the arm back to the left position (so the next station can catch the workpiece without clashing); the FREE button allows to move the manipulator arm from the left to the right position, in order to allow the feeder to push another workpiece without clashing; the ON LED indicates if the transfer unit is enabled, the READY LED indicates that the transfer unit is ready to accept a workpiece from the feeder (the left position is free) and the MOVED LED indicates that a workpiece has been moved to the right position.

The human operator must manually handle the synchronization between the units in the following way:

(1) push the FREE: the arm moves to the right position;
(2) wait for the READY LED to be lit to indicate the end of the operation;
(3) push the PUSH button: a workpiece is pushed out of the magazine;
(4) wait for the PUSHED LED to be lit to indicate the end of the operation;
(5) push the TRANSF button: the arm moves to the left position, takes the workpiece, moves the workpiece to the right position, leaves the workpiece and comes back to the left position;
(6) wait for the MOVED LED to be lit to indicate the end of the operation;
(7) command the next station to take the workpiece and go back to step 1.

The IEC 61131-3 program that controls the feeder unit is shown in Fig. 15a. The used language is Function Block Diagram (FBD) and the program basically connects the input buttons and output LEDs to the Feeder Function Block that implements most of the control logic. For simplicity, the behavior of the Feeder FB is shown as a state chart diagram in Fig. 15b that can be implemented with any IEC 61131-3 language. The EN input of the Feeder FB is used to

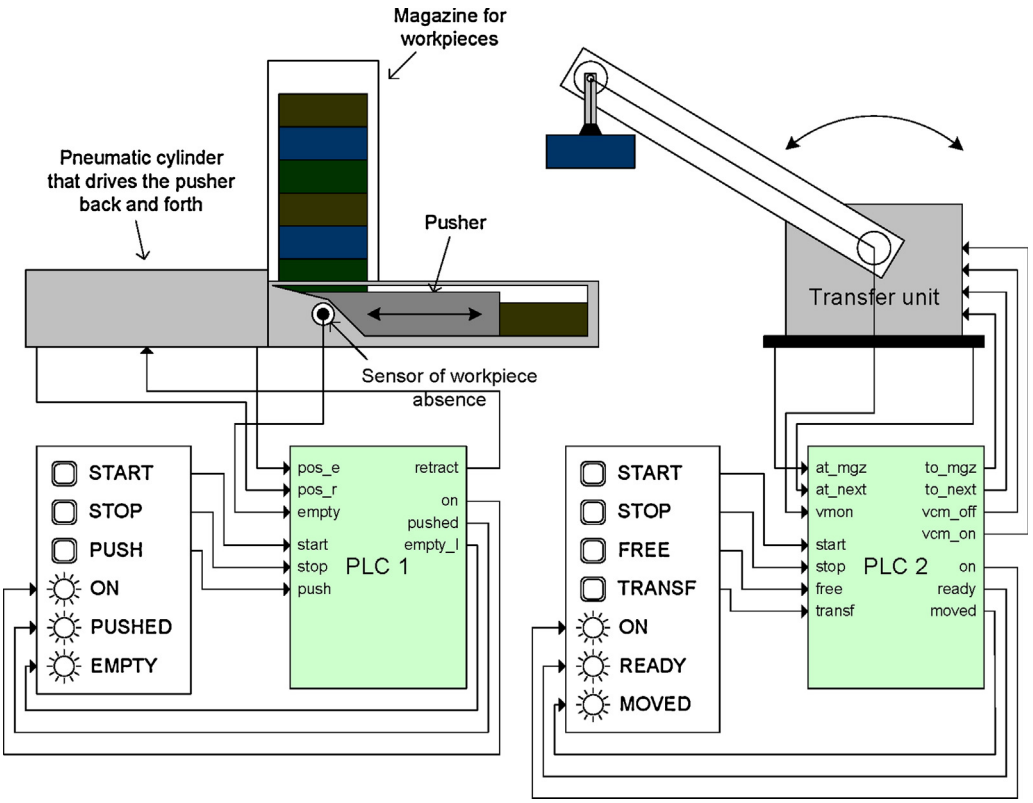


Fig. 14. The mechatronic units for workpiece storage [27]. The names of PLC input/output variables are mostly self-explanatory. An explanation of the less intuitive names follows: pos_e–position extended; pos_r–position retracted; vmon–workpiece sucked in; vcm_on/vcm_off–start/stop sucking a workpiece.

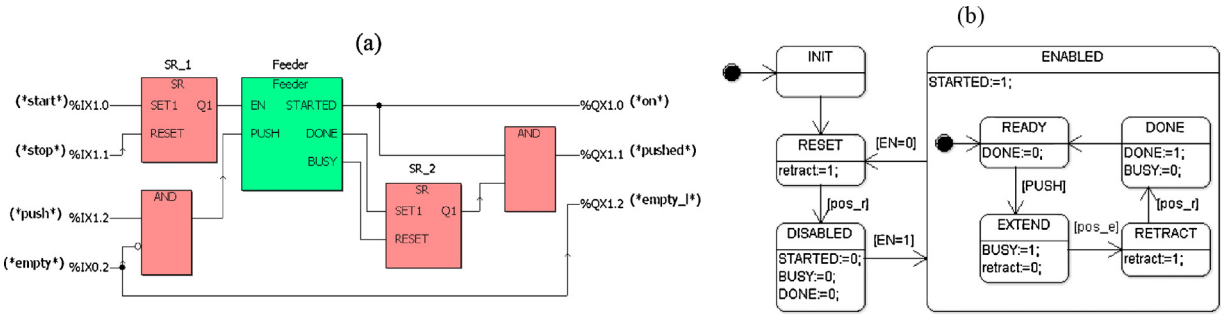


Fig. 15. IEC 61131-3 program that controls the feeder unit. (a) Main program in Function Block Diagram (pink/dark–standard FBs, green/clear–custom FBs) (b) State machine describing the behavior of the Feeder function block (Upper case variables refers to the Feeder FB input/outputs, while lower case variables refers to PLC 1 input/outputs shown in Fig. 14).

enable the function block. When the FB is enabled, the STARTED output indicates that the Feeder FB is enabled. When a rising edge is detected on the PUSH input, the BUSY output becomes active and a workpiece is pushed out of the magazine. When the operation is finished, the BUSY is deactivated (becomes off) and the DONE output becomes high (on) for one cycle.

The IEC 61131-3 program that controls the transfer unit is shown in Fig. 16a. Similarly to the feeder unit, the program connects the input buttons to the Transfer Function Block, whose behavior is shown in Fig. 16b. The EN input and the STARTED output work as it has already been seen for the Feeder FB. The FREE input allows to move the manipulator arm from the magazine to the right position, while the TRANSF input allows to transfer a workpiece from the magazine to the right position. When the FREE or the TRANSF inputs are activated, the BUSY output is activated until the requested operation is finished. Then the BUSY output is set to 0 and DONE is activated. The NEXT and MGZ outputs are updated according to the position of the manipulator arm.

5.3. A methodology for the integration based on the LE-INT architecture

The methodology proposed for the integration of existing systems in a more advanced distributed solution is based on (re) using IEC 61131-3 code for the implementation of the low-level control logic and using IEC 61499 for high-level integration control logic, by utilizing the communication facilities offered by the LE-INT architecture. The low-level IEC 61131-3 control logic deals with sensors and actuators and performs basic control tasks that are already developed in the existent systems. The high-level IEC 61499 control logic performs more complex tasks that involve interactions between different systems using functionalities and information provided by the low-level logic.

In particular, the steps for the integration of the systems are the following:

(1) Definition of the requirements of the distributed solution: in this step, the behavior of the integrated distributed system is defined. In case of integration of independent systems (as in this case study) this step requires some designing effort to individuate the additional activities of the automated solution, while in case of distribution of a centralized system, this step is straightforward since the requirements will not change.
(2) Definition of the PDEs and the Interaction Function Blocks: in this step, from the analysis of the IEC 61131-3 logic, functionalities to expose and information to be exchanged with the IEC 61499 logic are individuated; these functionalities and information are mapped to LE-INT PDEs and IFBs. In case of integration of independent systems (as in this case study) this step is quite simple, since each system has its own program. In case of distribution of a centralized system, this step requires more effort to decompose the program code. However, as already said, usually program code is organized in a modular way and may be easily decomposed.
(3) Implementation of IEC 61131-3 interface code: basic modifications are applied to the IEC 61131-3 code to add the PDEs.

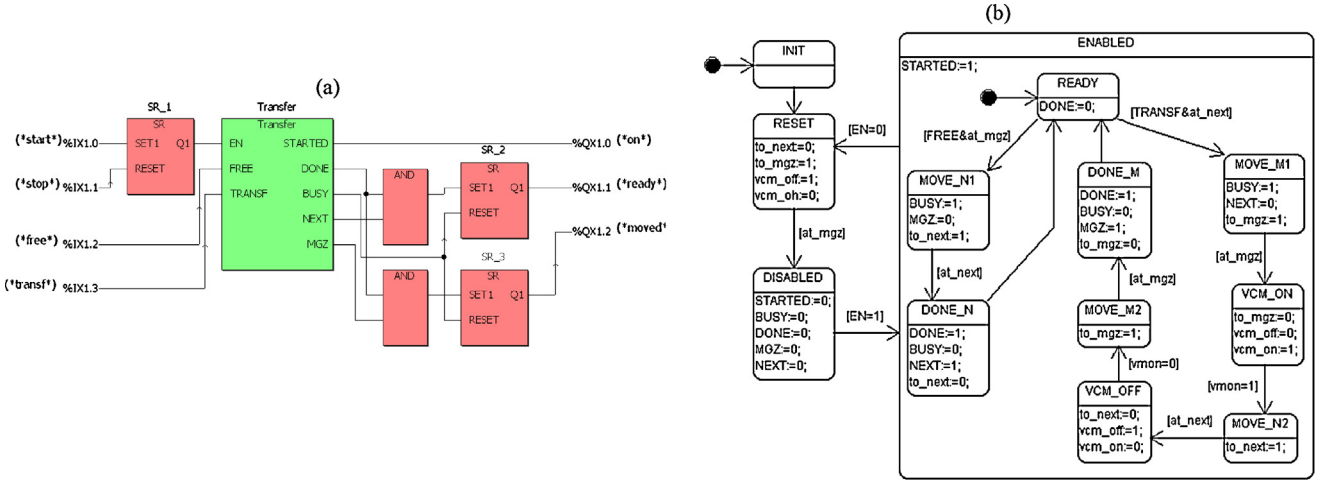


Fig. 16. IEC 61131-3 program that controls the transfer unit. (a) Main program in Function Block Diagram (pink/dark–standard FBs, green/clear–custom FBs) (b) State machine describing the behavior of the Transfer function block (Upper case variables refers to the Transfer FB input/outputs, while lower case variables refers to PLC 2 input/outputs shown in Fig. 14).

(4) Implementation of IEC 61499 control application: the high level distributed control application that fulfills the requirements defined in step 1 is developed, using functionalities offered by the IEC 61131-3 low-level layer.

5.4. Definition of the requirements of the distributed solution

In the first step, we define the behavior of the solution obtained integrating the feeder and transfer unit. This system allows the automation of the synchronization task presented in 5.2 and performed by the human operator. Two operative modes are defined: single or automatic mode. In single mode, the synchronization between the feeder and transfer units is managed by the distributed application, but the human operator is still necessary to handle the synchronization with additional successive stations of the production line, even though its work is reduced compared to the initial configuration. In this mode, the system waits for an acknowledgment from the operator after each workpiece is transferred. In automatic mode, all the synchronizations are handled by the distributed control application. In this case, the next station must be programmed so that it communicates information necessary to allow synchronization avoiding mechanical clashes between the stations. In this mode, no acknowledgment from the human operator is required.

The control panel of the new system has been redesigned to meet the new requirements. It has three buttons, a switch and two LEDs: START and STOP buttons allow to start or stop the feeder and the transfer units; the ACK button is used to confirm a workpiece transfer in single mode; the SINGLE switch allows to select single or automatic mode of operation; the ON LED indicates if the system is enabled and the MOVED LED is used in single mode to indicate that a workpiece has been transferred and the system waits for acknowledgment.

Fig. 17 shows a possible distributed solution for the control system. In this case, the precedent PLC configuration is maintained with few modifications to change the panel. The two panels are replaced with the panel designed for the new system. PLC 1 is connected to sensors/actuators relative to the feeder unit and also manages the input/outputs lines from the new control panel. PLC 2 is connected to sensors/actuators relative to the transfer unit. The two PLCs are now connected by a network that allows communication of synchronization information.

5.5. Definition of the interaction function blocks and implementation of IEC 61131-3 interface code

In the second step, we define the Interaction Function Blocks. An IFB for each unit is created. These are used to encapsulate (from the IEC 61499 point of view) the existing IEC 61131-3 code that manages the two units. In addition, a third IFB is created to interact with the IEC 61131-3 code that manages the command panel. We decided to implement the management of the control panel using IEC 61131-3 to show that the architecture can also be used in the design of mixed IEC 61131-3 and IEC 61499 systems from scratch.

Three IFBs are defined (Table 1):

- PI_FED: composed of the PDEs relative to the feeder unit;
- PI_TR: composed of the PDEs relative to the transfer unit;
- PI_PAN: composed of the PDEs relative to the panel.

The feeder unit IEC 61131-3 software layer implements the low-level control of the pusher. So, it exports to the IEC 61499 layer a PUSH service, that is used to command the feeder to push a new workpiece and it is realized with an Interlocked Control PDE. In addition, it communicates information about the emptiness of the magazine, the started status of the unit and it requires enable information from the IEC 61499 layer to enable/disable the unit.

So, the PI_FED has the following PDEs (Table 1):

- FED_EN: Programmed Data Acquisition PDE from IEC 61499 to IEC 61131-3 of the Boolean enable;

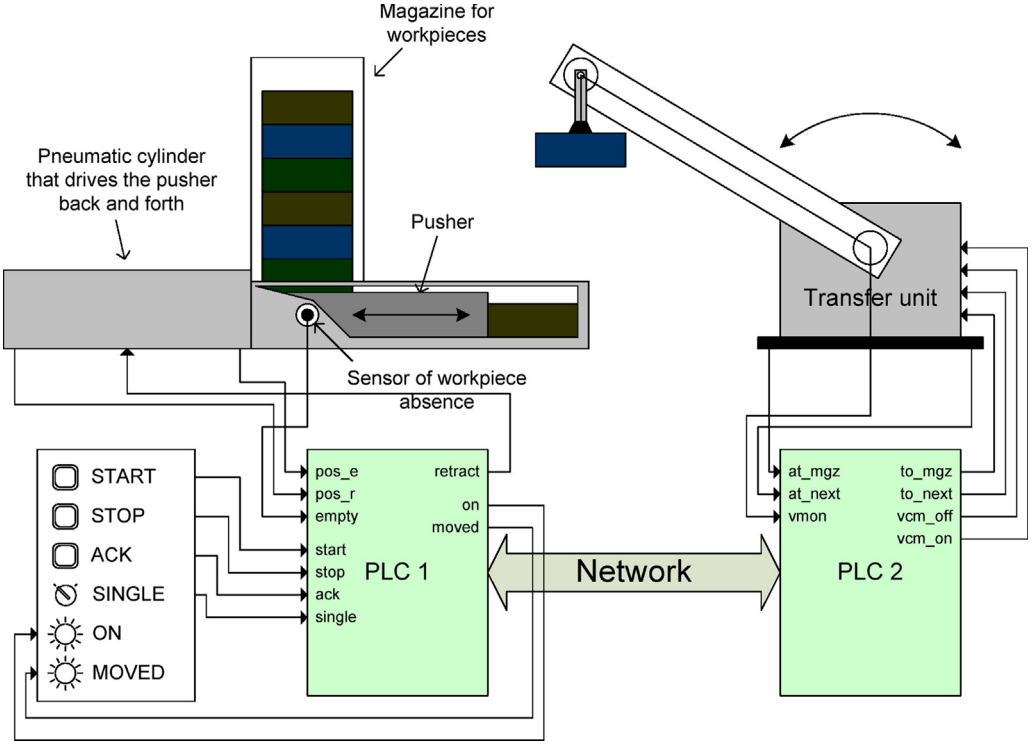


Fig. 17. A distributed scenario for the feeder and transfer units control.

Table 1
Organization of the IFBs used in the test case.

IFB	PDE	IEC 61131-5 FB	IEC 61131-5 FB Inputs/Outputs	IFB Data Inputs/Outputs
PI_FED	FED_EN	URCV	RD	EN
	FED_STARTED	USEND	SD	STARTED
	FED_EMPTY	USEND	SD	EMPTY
	FED_PUSH	RCV	–	–
PI_TR	TR_EN	URCV	RD	EN
	TR_STARTED	USEND	SD	STARTED
	TR_FREE	RCV	SD_0	MGZ
			SD_1	NEXT
	TR_TRANSFER	RCV	SD_0	MGZ
			SD_1	NEXT
PI_PAN	PAN_BTN	USEND	SD_0	START
			SD_1	STOP
			SD_2	ACK
			SD_3	SINGLE
	PAN_LED	URCV	RD_0	ON
			RD_1	MOVED

- FED_STARTED: Programmed Data Acquisition PDE from IEC 61131-3 to IEC 61499 of the Boolean started status;
- FED_EMPTY: Programmed Data Acquisition PDE from IEC 61131-3 to IEC 61499 of the Boolean empty status
- FED_PUSH: Interlocked Control PDE from IEC 61499 to IEC 61131-3 of the PUSH service that has no input/output parameters.

In Fig. 18a, the IEC 61131-3 code of the feeder unit is shown. The existing code (Figs. 15 and 16) is reused, and data lines that were previously connected to I/O of the control panel now interact with the darker/blue Function Blocks that constitute the PDEs executed. Note that Programmed Data Acquisition PDEs from IEC 61131-3 to IEC 61499 (FED_EMPTY, FED_STARTED) are performed only when the data change, so the CHGD FB is used to detect changes on the data to be sent. The CHGD code is shown in Fig. 18d.

The transfer unit IEC 61131-3 software layer provides the services FREE and TRANSFER that allow, respectively to command a free operation and a transfer operation. These operations are realized by Interlocked Control PDEs from IEC 61499 to IEC 61131-3. For error management, these PDEs carry two Boolean output parameters (MGZ and NEXT) that indicate the position of the manipulator after the operation is completed. In addition, similarly to the feeder unit, the transfer unit provides information about its started status and requires enable information from the IEC 61499 layer.

So, the PI_TR IFB has the following PDEs (Table 1):

- TR_EN: Programmed Data Acquisition PDE from IEC 61499 to IEC 61131-3 of the Boolean enable;
- TR_STARTED: Programmed Data Acquisition PDE from IEC 61131-3 to IEC 61499 of the Boolean started status;
- TR_FREE: Interlocked Control PDE from IEC 61499 to IEC 61131-3 of the FREE service with two Boolean inputs (MGZ and NEXT);
- TR_TRANSFER:: Interlocked Control PDE from IEC 61499 to IEC 61131-3 of the TRANSFER service with two Boolean inputs (MGZ and NEXT).

Fig. 18b shows the IEC 61131-3 code of the transfer unit. As for the feeder unit, the original code is left unmodified and the PDEs is added.

Finally, the panel control logic is designed from scratch. Following the proposed methodology, the input from the buttons and the outputs to the LEDs are managed by the PI_PAN IFB, that has the following PDEs:

- PAN_BTN: Programmed Data Acquisition PDE from IEC 61131-3 to IEC 61499 of the status of the buttons;
- PAN_LED: Programmed Data Acquisition PDE from IEC 61499 to IEC 61131-3 to drive the LEDs;

Fig. 18c shows the IEC 61131-3 code that simply connects the PLC input/outputs to the blue/darker FBs that performs the PDEs.

According to the hardware configuration shown in Fig. 13, the Feeder and the Panel programs are executed on PLC 1 and the Transfer program on PLC 2. Differently from Fig. 10, now PLC 1 and PLC 2 are connected via a network infrastructure.

5.6. Implementation of IEC 61499 control application

The application is organized in two blocks, one for controlling the feeder unit and the other one for controlling the transfer unit. In order to coordinate the operations between the two units, a simple protocol [24] is adopted. Fig. 19 shows a scheme of the protocol where the stations are modeled with an IEC 61499 function block: each unit of the production line communicates with the unit on the left and the one on the right, passing the variables ALLOW_LEFT and ALLOW_RIGHT and receiving LEFT_OK and RIGHT_OK. Before performing operations that could cause collision between mechanical parts, the controller must check the guard variable (LEFT_OK and RIGHT_OK) corresponding to the unit that may collide. These variables are set by the other units using ALLOW_RIGHT and ALLOW_LEFT. For example, if a station (called left station) wants to use a shared section that may collide with the station on its right (called right station), it must check its RIGHT_OK variable. This variable was previously set by the right station using its ALLOW_LEFT variable, to indicate that the shared section is not in use. While the left station uses the shared section, it must prevent the right station from accessing it, resetting ALLOW_RIGHT variable.

Fig. 20a shows the IEC 61499 control application. The dark/blue function blocks are the Interaction Function Blocks. The clear/green function blocks are IEC 61499 basic function blocks that implement the control application.

The FeederCtl FB implements the control of the feeder unit. It is connected with the IFBs PI_PAN and PI_FED and also sends the ALLOW_RIGHT variable to the TransferCtl FB and receives from it the RIGHT_OK variable for synchronization. The function block basically, when enabled, try to push workpieces from the magazine whenever possible, that is, when the transfer unit signals that it is ok to transfer (using the RIGHT_OK variable) and the magazine is not empty. The ECC (Execution Control Chart) of the FeederCtl function block is shown in Fig. 20b.

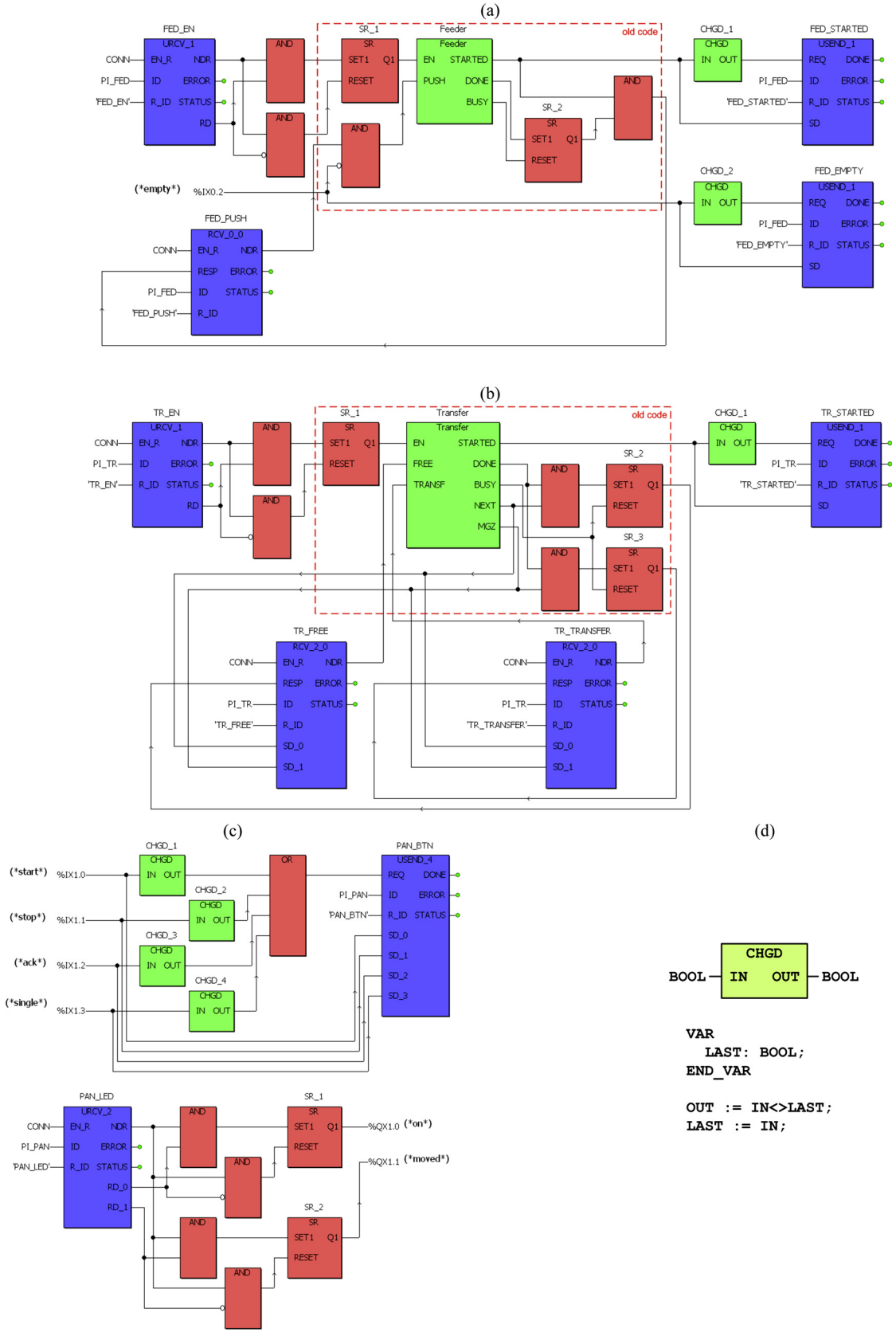


Fig. 18. IEC 61131-3 part of the code of the distributed system (pink—standard FBs, green—custom FBs, blue—interface FBs). (a) Program that controls the feeder unit, in Function Block (FB) Diagram. The legacy IEC 61131-3 code shown in Fig. 11 is reused. (b) Program that controls the transfer unit, in FB Diagram. The legacy IEC 61131-3 code shown in Fig. 12 is reused. (c) Program that controls the command panel. Written from scratch in FB Diagram. (d) Definition of the CHGD custom Function Block. Code is written in Structured Text. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

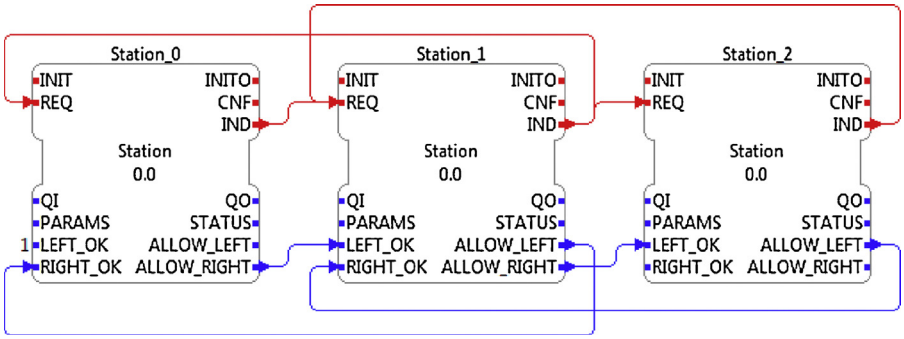


Fig. 19. A synchronization protocol [24] realized with IEC 61499 function blocks that represent the units of a production line.

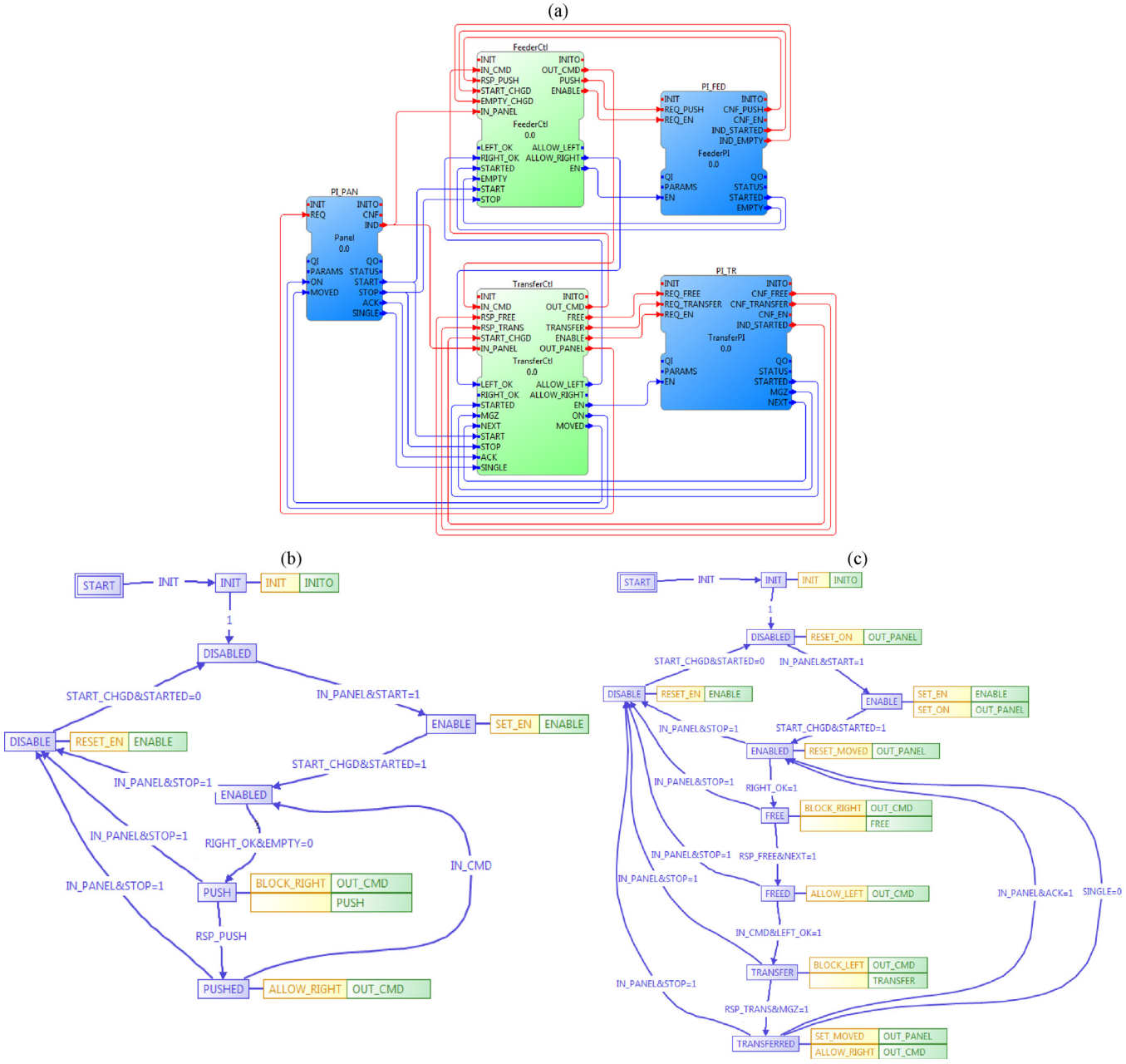


Fig. 20. IEC 61499 part of the code of the distributed system. (a) Distributed application. IFBs used to interact with IEC 61131-3 FBs are the darker/blue ones. (b) Execution control chart of the FeederCtl function block. (c) Execution control chart of the TransferCtl function block. Algorithms associated to the states are self-explanatory (e.g. SET_EN sets the EN variable to 1, while RESET_EN sets the EN variable to 0). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

In similar manner, the TransferCtl FB implements the control of the transfer unit. It is connected to the IFBs PI_PAN and PI_TR and it exchanges the ALLOW_LEFT and LEFT_OK variables with FeederCtl. In addition, an eventual station at the right of the transfer unit can implement the synchronization protocol sending the RIGHT_OK variable. Fig. 20c shows the ECC. When enabled, the station performs a FREE operation on the transfer unit (if an eventual unit to the right does not block it). When the FREE is completed, the FeederCtl block is allowed to execute the PUSH and the TransferCtl waits for the completion of the PUSH operation, that is the LEFT_OK variable is on. When this happens, the TransferCtl performs a TRANSFER operation and, if single mode is selected, it lights the MOVED LED and waits for the ACK from the control panel.

6. Implementation and evaluation

In this section, considerations about the possible implementations of LE-INT are presented and the details of the implementation used for the test case are described. In addition, we provide and discuss experimental results obtained by running the proposed test case and other implementations, based only on IEC 61499 or IEC 61131-3, in a simulated environment.

6.1. Chosen implementation

The test implementation of the architecture is based on the parallel execution of an IEC 61131-3 runtime and an IEC 61499 runtime. It has been realized on an industrial PC-based PLC environment with Microsoft Windows CE and also on a PC with Windows XP 32-bit.

We take a runtime environment for the IEC 61131-3 code [23] that 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 an open source runtime environment [23]. The SIFBs realizing the PLC interfaces have been implemented in the C++ language and included in the runtime.

Code modules belonging to PDEs communicate via shared memory.

IEC 61131-5 FBs were developed and installed 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.

SIFBs for the IEC 61499 part are designed as C++ classes. The behavior of a SIFB on the reception of events is defined by a particular function, while for the generation of SIFBs output events when a data exchange is initiated by IEC 61131-3, it has been introduced an external event handler thread that waits for data from IEC 61131-3.

6.2. Alternatives to the implementation

Different implementation strategies can be adopted to realize the proposed architecture, since it does not pose particular limits, as long as the control system (composed of different networked devices) is able to execute both IEC 61499 and IEC 61131-3 code and the IEC 61131-3 and IEC 61499 programs are able to communicate with each other via the defined interfaces and paradigms.

We have individuated three possible implementation approaches:

(1) The control system is composed of some IEC 61499 only devices and some IEC 61131-3 only devices. The PDEs realizing data exchanges between standards communicate through the network.
(2) Each device of the control system runs a runtime environment capable of executing both IEC 61131-3 and IEC 61499 programs, handling also the inter-standard communications.
(3) Each device of the control system executes in parallel the runtime environments for both standards and the data exchanges are realized by a software layer that uses inter-process communication techniques to transfer data between the runtime environments.

The first approach has the advantage that it does not require an heavy customization of the device, so it can be applied even to devices with a closed software architecture that does not permit to install a dedicated runtime for the other standard or does not permit to implement low-level custom function blocks. As long as the runtime provides basic communication function blocks for sending data through the network, the architecture can be implemented following this first approach. However this approach limits the flexibility of the proposed architecture, since each device is limited to execute code of a single standard. In addition, the execution of the code of different standards on different devices results in an increased inter-standard communication overhead [28] (data must be transmitted through the network) compared to the other approaches where all the code is executed on a single device.

The second approach is the most efficient of the three proposed approaches, since there is a single runtime that handles execution of modules of both standards, whenever the implementation of such environment requires a high development effort and specific competences about the software architecture of the devices. So this strategy can be implemented particularly by run-times developers.

The last approach is used in our implementation: it allows to execute programs of both standards on a single device, giving more flexibility than the first approach and usually reducing the inter-standard communication time overhead, since inter-process communication is faster than network communication and can exploits the benefits of CMP architectures [36]. Differently than the second approach, this one requires a low design effort since only the software layer that realizes the communication between the two runtimes must be implemented. As a drawback, the parallel execution of two runtimes can consumes more resources in terms of CPU usage and memory usage. The evaluation of such overhead is discussed in Section 6.4.

6.3. Evaluation environment

The aim of our evaluation was to characterize the performance of the proposed architecture in terms of: (i) the CPU and memory usage, (ii) the response time, and (iii) the PDE delay time.

The control software, the runtimes for the two standards, and the IFB and PDE were executed on real machines. In particular, the evaluation was performed on two hardware configurations. The first configuration (that will be referred as WinCE) is composed of two compact industrial PCs [44], each one with an ARM9 CPU at 520 MHz and 64 MBytes of RAM. The two devices use Windows CE as operating system and are connected to each other using an Ethernet network. The second configuration (that will be referred as Win32) is composed of two high-performance PCs: Intel Pentium 4 at 3 GHz with 2 GB RAM. The two PCs use 32-bit Windows XP as operating system and are connected to each other using an Ethernet network. In both configurations the devices were equipped with the IEC 61131-3 runtime and the IEC 61499 runtime. Only the mechatronic components of the system were simulated, and they were designed to generate the input changes as in the actual machines according to the timing constants shown in Table 2, derived from analysis of real systems. In performance

Table 2
Time constants used for configuring the simulation environment.

Time constant	Value (ms)
Feeder extension time	350
Feeder retraction time	250
Transfer move time (empty)	1450
Transfer move time (loaded)	1850
Vacuum on time	900
Vacuum off time	50

measures, their CPU and memory usage of these processes has not been considered.

For the tests, the control application was configured in SINGLE mode, so each workpiece transfer had to be confirmed by an ACK signal. The evaluation environment automatically sent the panel commands to start the transfer and feeder units and sent the ACK each time a workpiece was successfully transferred (MOVED LED active). The ACK was sent after a random amount of time, in order to decouple the pressure of the ACK button from the period of the control task. This workpiece transfer was repeated 100 times for each simulation run, while the evaluation environment collected the following performance data:

- response time: it is the time interval between the push of the ACK button (that indicates the system to transfer another workpiece) and the MOVED LED is lit (that indicates that the workpiece has been transferred);
- CPU usage: it is the percentage of CPU time dedicated to the execution of the runtimes and the control application;
- memory usage: it is the memory allocated by the runtimes for the execution of the control application;
- PDE delay time: it is the time interval between the instant in which a request arrives to a PDE (rising edge of the REQ input for IEC 61131-3 or REQ event for IEC 61499) and the instant when the data is available (rising edge of NDR output for IEC 61131-3 or IND event for IEC 61499).

6.4. Simulation results

The proposed architecture has been evaluated in different configurations and has been compared with different implementations. In particular, we considered the following implementations:

- LE-INT. It is the solution described in Section 5.6. This application has been tested on a single device (without network communications) and on two devices (networked solution). It has also been tested with different task periods for the IEC 61131-3 tasks: 10 ms, 50 ms and 100 ms.
- IEC 61499. It is a pure IEC 61499-based solution. The control application has been developed following the multi-layer distributed controllers approach [24]. This solution also has been executed on a single device or distributed on two devices.
- IEC 61131-3. It is a pure IEC 61131-3 solution. This is a centralized solution (described in [24] with a state machine) and it has been implemented only on a single device. The solution has been executed with different values of task period: 10 ms, 50 ms and 100 ms.

Each implementation has been tested on the hardware architectures described in the previous section: WinCE and Win32.

Fig. 21 shows the response time of the various configurations. The bars indicate the average recorded response time of the 100 runs of the simulation, while the line indicates the variance. The analysis of these results indicates that the pure IEC 61499 implementation has the best (lowest) response times and has the lowest variance. This is due to the better efficiency permitted by the event-driven model, opposed to the cyclic execution of IEC 61131-3. For the mixed and pure IEC 61131-3 configurations, the average response time and the variance depend on the chosen task period, since a larger period means an increased delay between the change of a PLC input and the relative processing. The use of the mixed approach over a pure IEC 61131-3 solution causes only a small increment in the response time (especially at 10 ms where the difference is practically insignificant).

The results showed also that there are not substantial differences between running the control application on one or two devices. This means that a single device is more than enough to satisfy the processing needs of application and also means that communications does not introduce significant delays to the control application (the communication overhead is compensated by the parallel execution on different devices, as the experiment are in a single core environment). Finally, because of the difference in computational power, Win32 response times are slightly lower than WinCE response times.

Fig. 22 shows various graphs of CPU and memory usage; we could not collect CPU usage data for Win32 configurations because the values were too small to ignore measure error (average CPU usage less than 0.1%). The results show that the IEC 61131-3 runtime resource consumption (both in terms of CPU and memory)

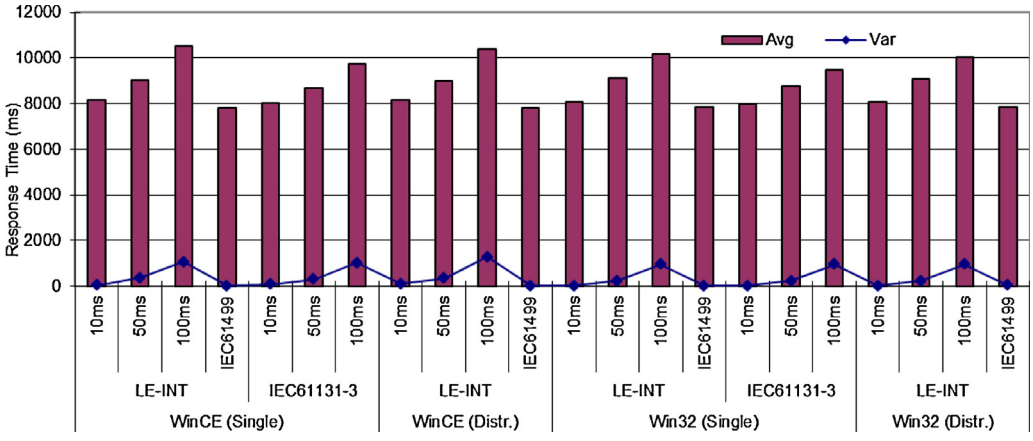


Fig. 21. Response times of the different tested configurations. The bars indicate the average response time while the line indicate the variance of response times.

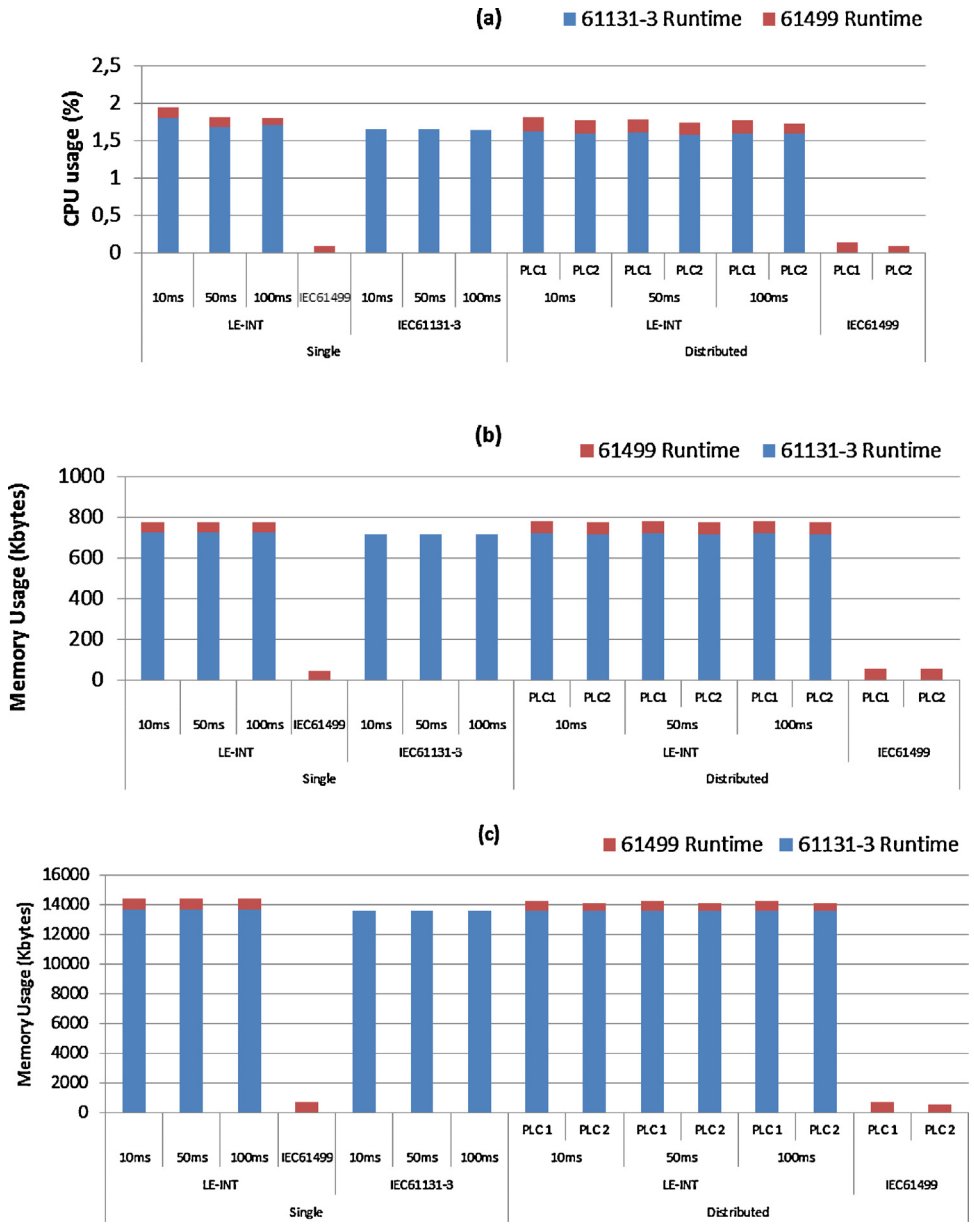


Fig. 22. Resource usage: (a) WinCE average CPU usage, (b) WinCE memory usage, (c) Win32 memory usage.

is far greater than IEC 61499 runtime resource consumption. From the CPU usage point of view, this can be explained by the fact that the IEC 61131-3 runtime uses a cyclic approach so it must keep a task scheduler running at high frequency to activate the tasks of the control application, while the 61499 runtime uses an event-driven approach where function blocks are executed only when needed. In particular, for the 61131-3 runtime, the CPU usage is determined in a large part by the runtime activity and only a very small part by the control application; in fact, even if the task period is changed or if the application is distributed on two controllers, the CPU usage remains basically the same. In the distributed approach, the CPU and memory usage of the 61499 runtime are increased, since it must also handle the data communication through the network. The contribution of the 61499 runtime in terms of resource consumption is negligible when compared to the 61131-3 one.

Summarizing, in terms of resource usage (memory and CPU), the proposed solution is similar to a pure IEC 61131-3 solution, it is greater than a pure IEC 61499 solution but it is not critical even on a low-end hardware configuration (less than 2% CPU usage in all the configurations). From the performance point of view, the faster solution is the pure IEC 61499 one, however it is possible to achieve similar performance in our solution by acting on the task period.

Fig. 23 shows the average delay times measured for the different PDEs. The delay have been measured in the LE-INT configuration with a single controller and only the PDE sufficiently stimulated during the simulation runs have been reported (FED_PUSH, TR_FREE, TR_TRANSFER, PAN_LED and PAN_BTN). The values have been normalized to the task period. The interlocked control PDEs perform data transfers in both directions, so they appear twice in the graph, once for sending the parameters and once for receiving the results. Data transfers from IEC 61499 to IEC 61131-3 have a delay, slightly smaller than 1 task period. This is due to the fact that data are actually available to the IEC 61131-3 runtime when the task is invoked, so even if the data transfer is instantaneous, the data reception is delayed to the next invocation. However, we always get a delay that is almost one period. This behavior can be explained by considering that in our particular

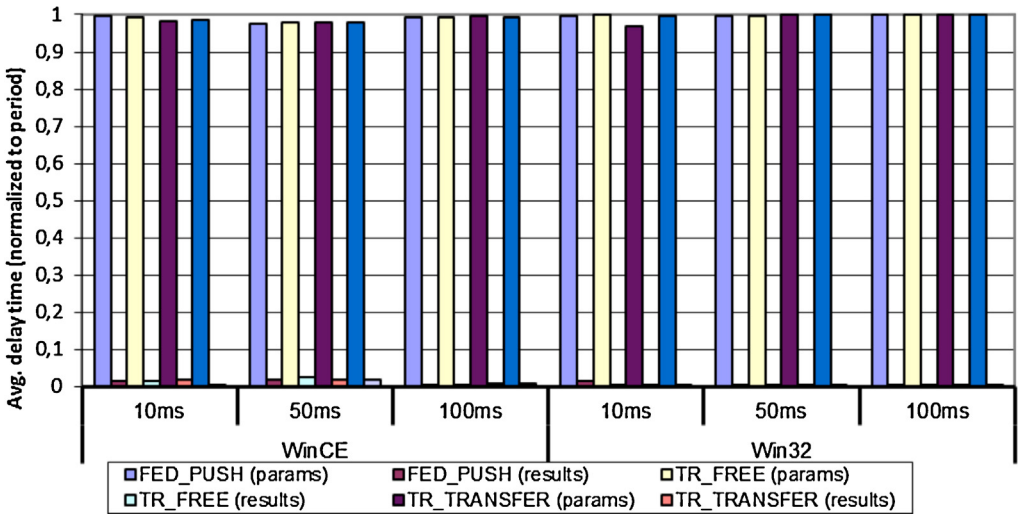


Fig. 23. Average delay time measured for the various PDEs. The values are normalized to the task period. Note that data transfers from IEC 61499 to IEC 61131-3 have a delay slightly smaller than 1 task period, while data transfers to the other direction (from IEC 61131-3 to IEC 61499) are almost instantaneous (less than a millisecond).

control application, every IEC 61499 event chain is generated by requests coming from the IEC 61131-3 layer, so the execution of IEC 61499 function blocks is synchronized with the task invocation.

The delay of communications from the IEC 61131-3 to IEC 61499, on the other hand, is very low (less than one millisecond), since the event-based execution model of IEC 61499 allows to process data as long as it is received without any further delay.

6.5. Discussion

As stated in Section 2, Zoitl et al. [12] proposed three different approaches to achieve harmonization between the two standards:

(1) parallel co-existence of the run-time of both the standards with a communication interface to provide interoperability;
(2) IEC 61131-3 based run time enhanced with IEC 61499 concepts and functionalities;
(3) IEC 61499 based run time enhanced with IEC 61131-3 concepts and functionalities.

Our proposed solution follow the first approach, while solution presented in [8,9,20,21] are solutions that follow the other two approaches. All the three approaches guarantee the integration and reuse of IEC 61131-3 code in an IEC 61499 solution. The main differences can be identified in the following points: the effort in realizing the run-time, the effort in realizing the inter-standard communication interfaces, and the performance overhead in terms of response time, CPU and other resources usage.

For our solution, the run-time support is not modified, so the only effort consists in coding the FBs model of our architecture, then the specific FBs of an application may be automatically generated from such model. The approaches 2 and 3 instead are more complex because they require the coding of the run-time support for a standard into the runtime of the other, and in fact, such complexity has been reduced in some cases by relaxing the full compatibility and compliance constraints. As for the effort in realizing the inter-standard communication interfaces, such effort is similar in all the three approaches, and it mainly concerns the specification of the data that must be exchanged. In conclusion, the effort to set-up our solution is minimal, and can be implemented with low additional resources, while some programming effort is required to realize integration. On the contrary, the effort to set-up the other two approaches is bigger, especially if they are devoted to the realization of systems that must run in critical environment, as it includes not only significant development effort but also significant testing effort.

From the performance point of view, data in Section 6.4 indicate that our solution, even when executed on a single machine, performs in a way similar to a pure IEC 61131-3 or IEC 61499 solution as for response time, while the CPU usage of the run-times, although higher with respect to the others, is not so significant (below 2%). As the proposed solution requires the parallel executions of two run-times, it represents the "worst case" among the three approaches, so we can expect that solutions based on the other approaches do not present significant differences with respect to our results.

As stated in the paper, the approaches 2 and 3 have been already implemented, although with some compliance issues. Our solution is a valid alternative, especially when there is the need to achieve fast solutions with low programming/economic effort.

7. Conclusion

In this paper, we propose an architecture for coexistence and interaction of IEC 61131-3 and IEC 61499 standard in the same control environment. The architecture allows to design distributed control logic using IEC 61499 while maintaining existing IEC 61131-3 software. In addition, new features can be added to the system using the most appropriate standard. In order to maintain compatibility with both standards and to reuse well known concepts and function blocks, the architecture is based on IEC 61499 service interface function blocks and IEC 61131-5 communication function blocks.

A methodology for the integration is presented via a case study, where the architecture is used to realize IEC 61499 distributed logic controlling two IEC 61131-3 units of a production line. The units previously operated independently and they needed a human operator for coordination. The case study shows how the proposed architecture and methodology can be used to integrate different systems maintaining the existent IEC 61131-3 control logic and using IEC 61499 to specify distributed control logic. This approach also improves the expansion possibilities of the system. In fact, if a new unit must be added to the distributed system, its control logic can be designed indifferently using IEC 61131-3 or IEC 61499.

The case study has been evaluated on various hardware configurations and it has been compared with a pure IEC 61499 solution and a pure IEC 61131-3 solution. From the resource (CPU and memory) consumption point of view, the proposed solution introduces an overhead that is larger than the pure IEC 61499 solution but comparable to the pure IEC 61131-3 solution; anyway, such overhead (derived from the coexistence of both IEC 61499 and IEC 61131-3 runtime environments) is relatively low even on limited hardware configurations (ARM CPU at 520 MHz), so if a pure IEC 61131-3 solution can be implemented on a particular hardware, it is also possible to implement a mixed solution with the proposed architecture on the same hardware. This means that the practices used for sizing a system based on a pure IEC 61131-3 solution can be utilized also for the proposed solution. From the performance point of view, the pure IEC 61499 solution is the fastest, but the proposed mixed solution can reach similar performance by acting on the task period. The inter-standard communication delay is smaller than a task period; this is comparable to the delay in detecting any other event in a cyclic execution model. As a summary, the advantages introduced by the LE-INT (derived from the opportunity of reuse, distribution and expansion) are not overwhelmed by performance drawbacks, so it represents a valid alternative to harmonization approaches proposed in literature, especially for low effort solutions.

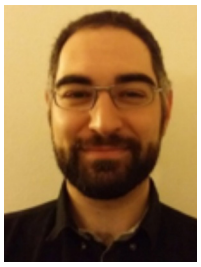
As for future works, we plan to characterize the real-time behavior of the proposed architecture, and to introduce support for debugging and testing. We plan also to apply the architecture to different application domains [22,43].

Acknowledgement

We are grateful to ISAC s.r.l. for the support provided in the setup of the hardware infrastructure.

References

[1] Programmable controllers – Part 3: Programming languages, IEC 61131-3 standard, 3rd ed., IEC, 2013.
[2] M. Buchner, Distributed computer control for industrial process systems characteristics, attributes, and an experimental facility, IEEE Control Syst. Mag. 2 (1982) 8–15.
[3] Function blocks – Part 1: Architecture, IEC 61499-1 standard, 2nd ed. 2012.
[4] V. Vyatkin, IEC 61499 as enabler of distributed and intelligent automation state-of-the-art review, IEEE Trans. Ind. Inf. 7 (2011) 768–781.
[5] P. Tait, A path to industrial adoption of distributed control technology, in: Third IEEE International Conference on Industrial Informatics, 2005, 86–91.
[6] M. Colla, A. Brusaferri, E. Carpanzano, Applying the IEC-61499 model to the shoe manufacturing sector, in: IEEE Conference on Emerging Technologies and Factory Automation, 2006, 1301–1308.
[7] K. Thramboulidis, S. Sierla, N. Papakonstantinou, K. Koskinen, An IEC 61499 based approach for distributed batch process control, in: Fifth IEEE International Conference on Industrial Informatics, 2007, 177–182.
[8] ⟨http://www.isagraf.com/⟩
[9] ⟨http://www.nxtcontrol.com/⟩
[10] K. Thramboulidis, IEC 61499 in factory automation, in: IEEE International Conference on Industrial Electronics, Technology and Automation (CISSE-IETA'05), Bridgeport, USA, 2005.
[11] K.H. Hall, R.J. Staron, A. Zoitl, Challenges to industry adoption of the IEC 61499 standard on event-based function blocks, in: Fifth IEEE International Conference on Industrial Informatics, 2007, 823–828.
[12] A. Zoitl, T. Strasser, C. Sunder, T. Baier, Is IEC 61499 in harmony with IEC 61131-3? IEEE Ind. Electron. Mag. 3 (2009) 49–55.
[13] Programmable controllers – Part 5: Communications, IEC 61131-5 standard, IEC, 2000.
[14] C. Gerber, H.M. Hanisch, S. Ebbinghaus, From IEC 61131 to IEC 61499 for distributed systems: a case study, EURASIP J. Embedded Syst. 2008 (1) (2008) 1–8.
[15] W. Dai, V. Vyatkin, Redesign distributed IEC 61131-3 PLC system in IEC 61499 function blocks, in: IEEE International Conference on Emerging Technologies and Factory Automation, 2010, 1–8.
[16] 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, in: IEEE International Conference on Emerging Technologies and Factory Automation, 2008, 369–376.
[17] G. Shaw, P. Roop, Z. Salcic, Reengineering of IEC 61131 into IEC 61499 function blocks, in: Eighth IEEE International Conference on Industrial Informatics, 2010, 1148–1153.
[18] M. Wenger, A. Zoitl, C. Sunder, H. Steininger, Transformation of IEC 61131-3 to IEC 61499 based on a model driven development approach, in: Seventh IEEE International Conference on Industrial Informatics, 2009, 715–720.
[19] Proposed Annex H. Available at: ⟨http://www.holobloc.com/stds/iec/sc65bwg7tf3/document/annexh_0.0.pdf⟩.
[20] V. Vyatkin, J. Chouinard, On Comparisons of the ISaGRAF implementation of IEC 61499 with FBDK and other implementations, in: Sixth IEEE International Conference on Industrial Informatics, 2008, 264–269.
[21] C. Sünder, A. Zoitl, J.H. Christensen, H. Steininger, J. Ritsche, Considering IEC 61131-3 and IEC 61499 in the context of component frameworks, in: Sixth IEEE International Conference on Industrial Informatics, 2008, 277–282.
[22] M. Annoni, A. Bardine, et al., 2012. A real-time configurable NURBS interpolator with bounded acceleration, jerk and chord error, Comput. Aided Des. 44 (6) (2012) 509–521, http://dx.doi.org/10.1016/j.cad.2012.01.009.
[23] A. Zoitl, T. Strasser, A. Valentini, Open source initiatives as basis for the establishment of new technologies in industrial automation: 4DIAC a case study, in: IEEE International Symposium on Industrial Electronics, 2010, 3817–3819.
[24] V. Vyatkin, M. Hirsch, H.-M. Hanisch, Systematic design and implementation of distributed controllers in industrial automation, in: IEEE International Conference on Emerging Technologies and Factory Automation, 2006, 633–640.
[25] M. Wenger, A. Zoitl, Re-use of IEC 61131-3 structured text for IEC 61499, in: IEEE International Conference on Industrial Technology, 2012, 78–83.
[26] S. Campanelli, P. Foglia, C.A. Prete:, Integration of Existing IEC 61131-3 systems in an IEC 61499 distributed solution, in: ETFA 2012—IEEE International Conference on Emerging Technology & Factory Automation, 17–21st September 2012, Krakow, Poland, IEEE CS, Los Alamitos, CA, 2012.
[27] V. Vyatkin, IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design, ISA, USA, 2006.
[28] A. Bechini, P. Foglia, C.A. Prete, Use of a CORBA/RMI gateway: characterization of communication overhead, in: Proc. of the Third Inter. Work. on Software and Performance, ACM, New York, NY, USA, (2002), pp. 150–157, http://dx.doi.org/10.1145/584369.584392.
[29] A. Di Stefano, C. Santoro, A3M: an agent architecture for automated manufacturing, Softw.: Pract. Exp. 39 (2) (2009) 137–162.
[30] K. Bengtsson, B. Lennartson, O. Ljungkrantz, C. Yuan, Developing control logic using aspect-oriented programming and sequence planning, Control Eng. Pract. 21 (2013) 12–22.
[31] C. Yang, V. Vyatkin, Transformation of Simulink models to IEC 61499 Function Blocks for verification of distributed control systems, Control Eng. Pract. 20 (12) (2012) 1259–1269.
[32] D. Hästbacka, T. Vepsäläinen, S. Kuikka, Model-driven development of industrial process control applications, J. Syst. Softw. 84 (2011) 1100–1113.
[33] J. Peltola, J. Christensen, S. Sierla, K. Koskinen, A migration path to IEC 61499 for the batch process industry, in: Fifth IEEE International Conference on Industrial Informatics, 2007, 811–816.
[34] K. Thramboulidis, D. Perdikis, S. Kantas, Model driven development of distributed control applications, Int. J. Adv. Manuf. Technol. 33 (4) (2007) 233–242.
[35] M. Bonfé, C. Fantuzzi, C. Secchi, Design patterns for model-based automation software design and implementation, Control Eng. Pract. 21 (11) (2013) 1608–1619.
[36] P. Foglia, et al., Exploiting replication to improve performances of NUCA-based CMP systems, ACM Trans. Embed. Comput. Syst. 13 (March) (2014), http://dx.doi.org/10.1145/2566568, 3s, Article 117; 23 pages.
[37] C.T. Papadopoulos, M.E.J. O'Kelly, M.J. Vidalis, D. Spinellis, Analysis and Design of Discrete Part Production Lines, Springer, London, New York, 2009.
[38] T. Hussain, G. Frey, Migration of a PLC controller to an IEC 61499 compliant distributed control system: hands-on experiences, in: IEEE International Conference on Robotics and Automation, 2005, 3984–3989.
[39] M. Hirsch, C. Gerber, H.-M. Hanisch, V. Vyatkin, Design and implementation of heterogeneous distributed controllers according to the IEC 61499 Standard—a case study, in: Fifth IEEE International Conference on Industrial Informatics, 2007, 829–834.
[40] M.N. Rooker, C. Sünder, T. Strasser, A. Zoitl, O. Hummer, G. Ebenhofer, Zero downtime reconfiguration of distributed automation systems: the εCEDAC approach, in: Third International Conference on Industrial Applications of Holonic and Multi-Agent Systems, 2007, 326–337.
[41] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritshow, G. Ulsoy, H. Van Brussel, Reconfigurable manufacturing systems, Ann. Int. Inst. Produc. Eng. Res. 48 (2) (1999) 527–539.
[42] V. Vyatkin, The IEC 61499 standard and its semantics, IEEE Ind. Electron. Mag. 3 (4) (2009) 40–48.
[43] P. Foglia, C.A. Prete, M. Zanda, An inspection system for pharmaceutical glass tubes, WSEAS Trans. Syst. (2015), in press.
[44] ⟨http://www.isacsrl.eu/uploads/media/Technical_Details_U_01.pdf⟩
[45] W. Dai, V. Dubinin, V. Vyatkin, Migration from PLC to IEC 61499 using Semantic Web Technologies, IEEE Trans. Syst. Man Cybern., A: Syst. Humans (2013), in press.
[46] A. Bartoli, P. Corsini, G. Dini, C.A. Prete, Graphical design of distributed applications through reusable components, Parallel Distrib. Technol.: Syst. Appl., IEEE 3 (Spring (1)) (1995) 37, 50.



Stefano Campanelli received his PhD in Information Engineering from University of Pisa in 2013. His main research interest were optimization algorithms for CNC machining and distributed control and automation. From 2014, he works for a multinational corporation in finance and trading automation.

Pierfrancesco Foglia is currently assistant professor at the Department of Information Engineering, University of Pisa, Italy. He received his PhD in Computer Engineering from the University of Pisa, where he teaches Informatics, and Industrial Applications. His research interests include computer architecture, low power computer design, coherence protocols, embedded systems and industrial informatics. He has leaded several research projects on embedded systems and industrial informatics funded by companies, and participates in EU and Italian Government research projects on high performance multiprocessor systems. He has advised several PHD students on computer architecture and industrial informatics topics. He is member of the NoE HIPEAC, and has served as General Chair of the HIPEAC 2010 Conference.

Cosimo Antonio Prete is currently Professor of Computer Architecture and Industrial Applications at the the Department of Information Engineering, University of Pisa, Italy. He has been Coordinator of the graduate degree program in Computer Engineering at the University of Pisa, Italy, *Rector's Advisor* for Innovative Training Technologies at the University of Pisa and Member of the Scientific Board of the PhD course in "Computer Science and Engineering" at the IMT institute for Advanced Studies, Lucca, Italy. He has extensive research record covering computer architectures, optimization techniques and development environments for embedded systems, development environments for parallel and distributed systems, high performance multi-core systems, coherence protocols for cache memories, design tools for hardware-software codesign, methodologies and tools for ensuring web usability. He has more than 100 technical publications in these and related research areas.