Hindawi Publishing Corporation
Journal of Engineering
Volume 2013, Article ID 638521, 9 pages
http://dx.doi.org/10.1155/2013/638521

Research Article

IEC 61499 as an Enabler of Distributed and Intelligent Automation: A State-of-the-Art Review—A Different View

Kleanthis Thramboulidis

Department of Electrical and Computer Engineering, University of Patras, Eratosthenous Street 6, 26504 Rio Patras, Greece

Correspondence should be addressed to Kleanthis Thramboulidis; thrambo@ece.upatras.gr

Received 15 August 2012; Revised 28 November 2012; Accepted 19 December 2012

Academic Editor: Piero Castoldi

The industrial and research activities around the IEC 61499 architecture for distributed automation systems are discussed by Vyatkin, (2011). Research results related to the design of this kind of systems as well as to the execution of IEC 61449 on embedded devices are reviewed. It is claimed that IEC 61499 has been developed to enable intelligent automation through the distribution of software components and that the industry will benefit through its adoption from the promise of the intelligence automation research results. In this paper, several observations are presented on the arguments that are used to prove that IEC 61499 is a solid technology for industrial automation systems development. Portability, interoperability, code modularity, reusability, and reconfigurability as well as determinism and the event-driven execution of IEC 61499 are discussed.

1. Introduction

In [1], the industrial and research activities around the IEC 61499 architecture for distributed automation systems are discussed, and IEC 61499 is promoted as an enabler of distributed and intelligent automation. It is claimed that the function block (FB) model, that this standard introduces, provides a framework for the distribution of intelligence automation in software components that can be freely distributed across networked devices. Execution semantics of the IEC 61499 Function Block model are also discussed and compared with the widely accepted by industry IEC 61131. Based on the presented reasoning, it is claimed in conclusion ([1, Section IX]) that the "extensive research effort in the past decade produced a very solid collection of results to be taken and used by industry" and that today these "first results can be observed in the market" allowing the users to enjoy the numerous "design time benefits" of the IEC 61499 tool chains for system-level design and implementation of automation systems." Unfortunately, the review and the discussion that lead to this conclusion are based on many unsubstantiated claims. For example, the so-called "event-driven execution" is considered as one of the key benefits of 61499 and as an enabler of portability and distributed application development [1, Section II-C]. Moreover, as claimed in [2] ([1, Reference 24, Section I]), the IEC 61499 standard has "emerged in response to the technological limitations encountered in the currently dominating" standard IEC 61131, which is based on the "cyclic scan" model that "is severely inadequate to meet the current industry demands for distributed, flexible automation systems." However, as it comes from Section 4, almost all the mentioned tools of IEC 61499, including the one used as proof of success, are not based on the event-driven paradigm. It is also claimed that the event-driven paradigm is implemented by FBDK and FORTE. But, it is well known that FBDK is based on the model of synchronous method call.

In this paper, a number of observations are presented on the arguments that are used in [1] to prove that IEC 61499 is a solid technology for industrial automation systems development. Portability, interoperability, code modularity, reusability, and reconfigurability as well as determinism and the event-driven execution of IEC 61499 are discussed. The remainder of this paper is organized as follows. In Section 2, a brief introduction to the IEC 61131 and IEC 61499 software models is given. In Section 3, observations on the example

of application used in [1], and the commercial tools used for its development are presented. In Section 4, the key features of IEC 61499 as defined in [1] are discussed. In Section 5, observations on determinism and the event-driven paradigm are presented. Section 6 presents a few other observations, and the paper is concluded in the last section.

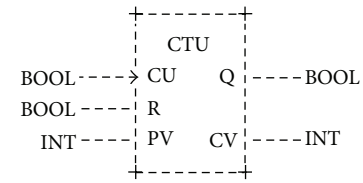

Figure 1: Graphical representation of the interface of the up-counter FB.

2. Background and State of the Art

Software for controlling industrial processes or machines is commonly executed on Programmable Logic Controllers (PLCs), which are digital computers specifically designed for the use in industrial environments, that is, environments with extended temperature ranges and high electrical noise and vibrations. PLCs provide advanced functions, among which analog monitoring, control, and high speed motion control and communication over networks. Moreover, PLCs satisfy requirements imposed by hard real-time applications in this domain. Today, even though PC-based controllers and distributed control systems (DCSs) can be used in this domain, the PLC is "still used for control in almost every manufacturing facility in the world" and is considered as one of the "most important automation inventions of all time." [3].

The IEC 61131 standard [4], the first version of which was published in 1993, was the first attempt of the International Electrotechnical Commission (IEC) to define a standard language for writing software for PLCs. More specifically, as claimed in [4], its scope is to specify "syntax and semantics of programming languages for programmable controllers." In practice, 61131 standardized several languages that were already used in this domain. Two of them, that is, IL (instruction list) and ST (structured text), are textual languages, and the other two, that is, LD (ladder diagram) and FBD (function block diagram), are graphical. Moreover, a higher level programming language, the sequential function chart (SFC), is defined for structuring the internal organization of programmable controller programs and function blocks. The FBD language is the one that better matches with the object-oriented and the model-driven development paradigms in software development and is widely used in industry.

The IEC 61131-3 standard has been adopted by the industry and is widely used by control engineers in specifying the software part of systems in the industrial automation domain. It was recognized worldwide as the standard for programming and configuring industrial control devices. To this direction, the contribution of PLCOpen [5], which provides an infrastructure for the portability of the 61131 models through the use of the PLCopen XML, was very important. However, IEC 611131 imposes several restrictions in the development process of today's complex industrial automation systems. As claimed in [6]: the standard "provides a small basis for common modelling of control programs, but platforms and tools are not able to interoperate." To address these restrictions as well as the new challenges in the development of today's complex industrial automation systems, the IEC has assigned to its Technical Committee 65 (IEC TC65) the task of developing a common model for the use of FBs [7]. The result of this activity was the IEC 61499 standard [8], which is considered as an extension of the IEC 61131 FBD language. At the same time, to overcome these restrictions, several research groups, for example, [9–11], have proposed approaches that exploit best practices from the desktop application domain. Object orientation, component-based development, and model-driven engineering are among these widely accepted best practices. Standardization of 61131 is also moving to this direction [12]. The IEC 61131 working group has proposed for voting a new version of the standard with the object-oriented (OO) extension being the main feature of this new version. Key points of this extension are as follows:

(a) the CLASS construct,

(b) the support for interfaces through the INTERFACE keyword,

(c) the extension of the FB type to include methods defined with the construct METHOD,

(d) the support of inheritance with the keyword EXTENDS.

Authors in [13] claim that the FBD language of the IEC 61131 has already introduced in the industrial automation domain, at least at the specification level, the basic concepts of the OO paradigm. It is also claimed that any attempt not only to use the IEC 61131 in model-driven engineering, but also to extend it to provide full support for OO should be based on the OO view of the existing standard.

2.1. IEC 61131 FBD. The key construct of the FBD language is the function block. Functions may also be used as behavioral units in FBDs. Behavioral units, that is, Functions Blocks and functions are interconnected in function block diagrams (FBDs) to specify the behavior of the controlling software. Interconnections are based on one type of connection which does not specify the type of information that flows from one end to the other, that is, there is no distinction between data flows and event flows. The direction of the flow is defined from the convention that outputs are shown on the right side of the FB, while inputs are on the left side. Figure 1 presents the interface of the up-counter 61131 FB in graphical notation, while Figure 2 presents its body in textual format, both as defined in IEC 61131 [4]. The body of an FB can also be presented in graphical notation using FBD. In this case, instances of behavioral units are interconnected to form function block diagrams to graphically specify the behavior captured by the body. For example, for an instance of the CTU FB, there will be up to three input connections and up to two output ones in an FBD.

```
IF R
THEN CV := 0;
ELSIF CU AND (CV < PVmax)
  THEN
  CV := CV + 1;
END_IF;
Q := (CV <= PV);
```

Figure 2: FB body of the up-counter FB in ST language.

According to the IEC 61131, a controller is considered as an aggregation of PLCs and the controlling software. A PLC is considered as an aggregation of CPUs, a power unit, I/O modules, and communication modules. The software model of IEC 61131 defines two types of elements. The first type consists of elements which are programmed using the languages defined by the standard; programs and function block types are among these elements. The second type consists of configuration elements among which configurations, resources, tasks, global variables, access paths, and instance-specific initializations. These elements are defined to support "the installation of programmable controller programs into programmable controller systems" [4]. The controlling software is organized as a composition of program organization units (POUs). The construct of POU is used to refer to the constructs of program, function block (FB), or function and is defined as an aggregation of a declaration part and a code part, as shown in Figure 3, which depicts the key elements of program organization. The same figure also captures the allowed calls between POUs.

2.2. IEC 61499. The IEC 61499 is considered to have emerged in response "to the technological limitations encountered in the currently dominating standard of IEC 61131" [14]. IEC 61499 extends the function block model of IEC 61131 to exploit best practices from the desktop application domain. More specifically, it adopts basic concepts of object orientation, and it describes an architectural model for the distributed applications in automation in a very generic way. A control application is defined as an aggregation of interconnected Function Block instances. The FB, which is the main building block of the IEC 61499 FB model, is defined as a design level construct to encapsulate industrial algorithms and the data that these algorithms operate on. It consists of a head and a body; the head of the FB type is used to capture the dynamics, and the body is used to capture the functionality, as shown in Figure 4. The head is connected to the event flows and the body to the data flows. The functionality of the function block is provided by means of algorithms, which process inputs and internal data and generate output data.

The sequencing of algorithm invocations is defined in the FB type specification using a variant of statecharts called execution control chart (ECC). An ECC consists of EC states, EC transitions, and EC actions as shown in Figure 5. An EC state may have zero or more associated EC actions, except from the initial state that may have no associated EC actions. An EC action may have an associated algorithm and an event that will be issued after the execution of the algorithm. EC transitions are directed links that represent the transition of the FB instance from a state to another. An EC transition is fired when the associated Boolean expression becomes true. The FB enters the target state, executes the associated algorithms, and issues the corresponding events.

Several research groups, for example, [15–18], have proposed approaches to extend IEC 61499 and overcome identified shortcoming of the standard.

3. Observations on the Example Application and the Tool

The author in [1] claims that industrial applications of commercial IEC 61499 compliant tools and platforms confirm their "benefits in terms of much improved design performance." ISaGRAF is considered as a characteristic example of such a tool. An experimental shoe manufacturing factory [19] ([1, Reference 5]), that has been developed using this tool, is used as a proof of the author's claims. More specifically, code modularity, reusability, and reconfigurability of the resulting system are the features that are advertised as practical benefits of applying IEC 61499 in this case study and in general in the industrial automation domain. However, the author does not refer on the features of the ISaGRAF (http://www.isagraf.com/) implementation of IEC 61499 that allow the development of systems with these features. It should be noted that ISaGRAF implements the IEC 61499 FBD based on the FB concept of IEC 61131 [20, Section IV] ([1, Reference 26, Section IV]).

3.1. Code Modularity. The paper uses the function block diagram shown in Figure 6 ([1, Figure 2]) to present code modularity in terms of function block instances and the ability of the designer to define the application as an aggregation hierarchy and to define the lowest level FBs by using execution control chart (ECC) and traditional PLC languages. However, a similar level of code modularity is obtained using the IEC 61131 FBs. The behavior of a program may be defined by a Function Block Diagram (FBD), and the behavior of the FB instances used in the FBD can also be defined using FBDs resulting to an aggregation hierarchy similar to the one described in [1] for IEC 61499. Moreover, the behavior of low level FBs can be defined in IEC 61131 in a similar way to the ECC by using the SFC [11]. This similarity is exploited by ISaGRAF that does not provide support for ECC and implements the IEC 61499 FBD based on the FB concept of IEC 61131 [20, Section IV]. Specifically, as mentioned in [20, Section III], "In ISaGRAF, ECC is implemented using sequential function chart (SFC) programming language" and "FB ECCs are invoked in a fixed order set prior to execution. Presence or absence of events does not influence the invocation." It is also known that ISaGRAF regarding the control behavior of IEC 61499 FB has "a bit different semantics, in accordance with the IEC61499 ECC," "although using the syntax of the IEC61131 standard," [20, Section III].

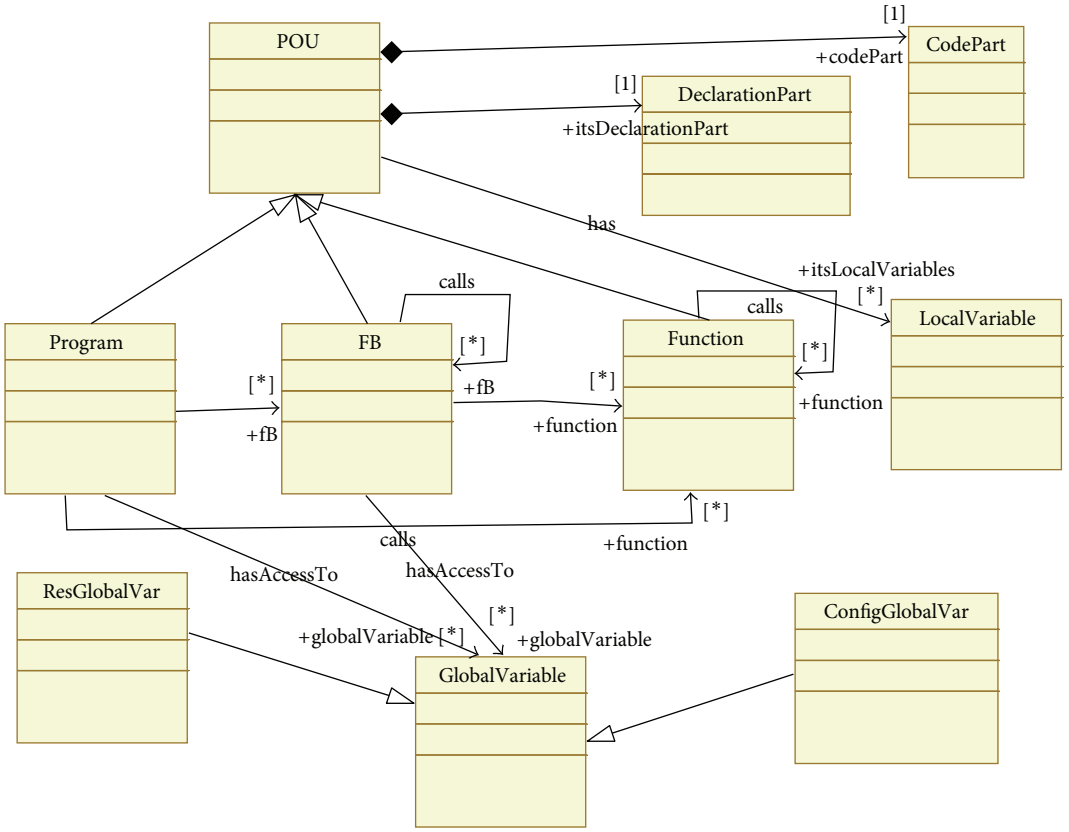

Figure 3: Key elements of the IEC 61131 software model regarding the organization of programs.

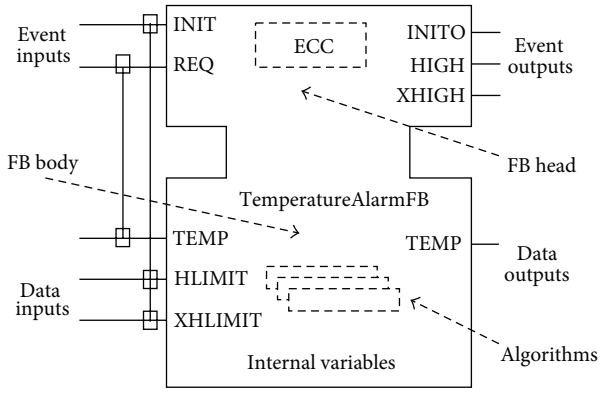

Figure 4: Graphical representation of the FB-type design level construct.

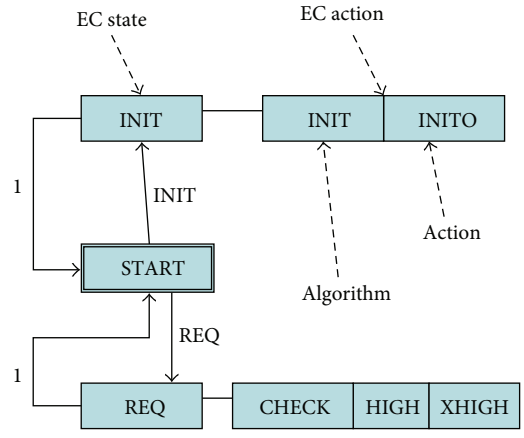

Figure 5: A simple execution control chart (ECC) to define the dynamic behaviour of FB instance.

3.2. Reusability and Reconfigurability. Regarding reusability, no arguments are given in [1] to prove that ISaGRAF through its 61499 FB implementation improves reusability compared to the 61131 implementation. The industry has exploited reusability, for many years now, through the reuse of the IEC 61131 FBs. Moreover, when the PLCOpen XML representation is adopted, portability of the design specifications is also a benefit, as in the case of IEC61499. Moreover, as argued in [11], instead of what is widely believed, IEC 61131 has already introduced in the industrial automation domain, at least at the specification level, the basic concepts of the OO paradigm. The FB may be considered as a special kind of class not only with several restrictions, but also with extensions. In a similar way to the class, it has a name; it defines the state of its instances using a set of local variables, declared either textually or graphically, and it defines the behavior of its instances through its body. All these, provide quite similar benefits for reusability as that of the IEC 61499 FB.

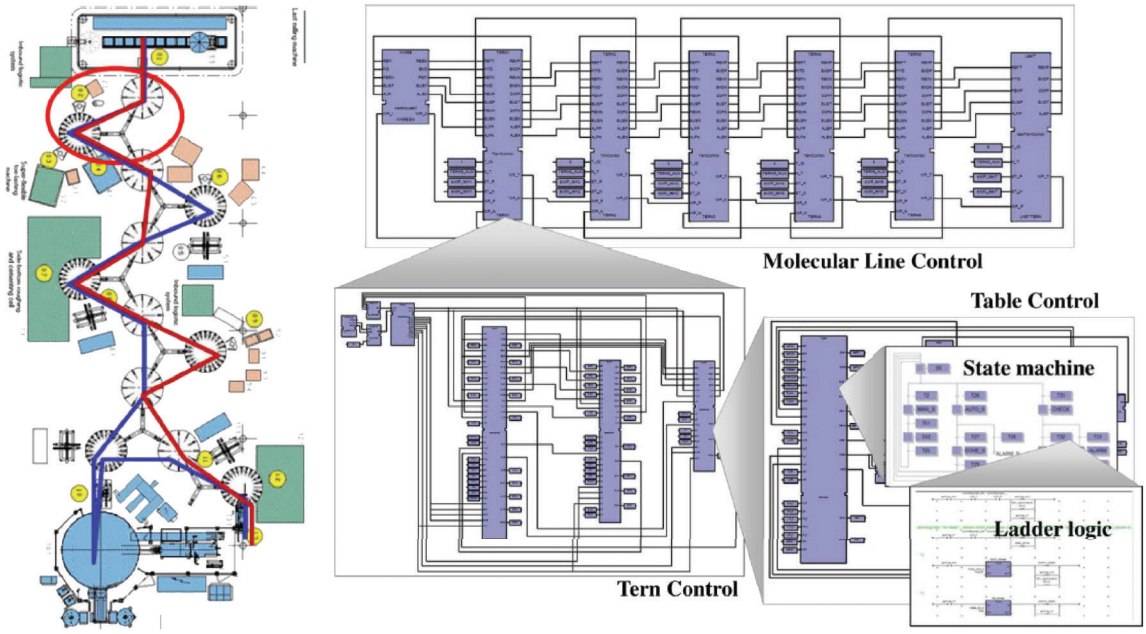

FIGURE 6: (a) Product flows through the "molecular line" and (b) function block control enabling the flexibility of the "molecular line" [1].

Regarding reconfigurability:

(a) it is not mentioned if it refers to run-time reconfigurability,

(b) no argument is given to prove that the ISaGRAF implementation improves reconfigurability compared to the one supported for the IEC 61131 implementation. It is quite interesting to see how an IEC 61131 run-time environment may be used to support run-time reconfigurability for IEC 61499 applications in a better way than for the IEC 61131 ones.

4. Observations on the Features of IEC 61499

It is claimed in [1] that one of the main features of IEC 61499 is the duality of the FB language construct. This is based on the argument that its main design artifact, that is, the Function Block, has been extended from the subroutine-like structure of 61131 FB to the process-like abstraction. However, in this way, the FB, that is a design-time artifact, is compared with implementation constructs such as subroutines and processes. It is well known that a design-time artifact such as the FB, can be implemented as a process, and this is also valid for the IEC 61131. Moreover, the event interface of the 61499 FB is presented as a successful mean of modeling interprocess message-based communications. However, the objective of design-time artefacts is not to represent implementation time mechanisms at the design specification. It is common for mechanisms of lower levels of abstraction, such as the implementation ones, to implement the higher level constructs such as the event interface of FBs or the messages in UML sequence diagram that match their semantics. Observations on the event interface of IEC 61499 are presented in the next section.

Another main feature, according to [1], is its duality of the application design, that is, being both model and implementation. But this is valid for every design specification for which an execution engine can be developed. The prerequisite for this is the existence of very well defined execution semantics, such as the ones of Harel Statecharts [21]. But this is one of the big problems of IEC61499, that is also reported in [1, Section IV-A]. It should be mentioned that the tool, which is used by the author as proof of the success of IEC 61499, transforms the design specification to an IEC 61131 executable model that is executed on the IEC 61131 run-time environment. Moreover, it is also claimed that the 61499 architecture supports, in the same design, combination of several diagram types such as "block diagrams, state charts, and ladder logic." However, it should be noted that these diagrams are used to capture structural and behavioral aspects of the application in low level of abstraction, and this is done in an unsatisfactory way [22]. It is also claimed that "a particular executable FB configuration needs to include platform-dependent service interface function blocks, which hinders the portability." However, an executable may include service interface function blocks (SIFBs), only in the case that SIFBs have been appended, even automatically, in the design specification, during deployment, to implement the event connections between FBs that are allocated to different nodes. Moreover, since SIFBs should be implemented on top of a middleware to ensure interoperability, portability is not hindered. Portability is a feature that can be considered before and after deployment.

The third main feature is what the author calls "event-driven execution." Since it is not clear, at least in [1], Section 2,

what the author means by this term, we consider the following two alternatives: (a) it refers to the activation of the behavior defined by an application in response to an external event and (b) the implementation of event connections between FB instances. In the first case, an IEC 61499 application may be activated either based on time (time triggered), in the case of periodic activation, for example, the use of E_CYCLE FB, or upon the occurrence of an event (event triggered). Both activation mechanisms are supported by IEC 61131 through the use of the task, which is one of its execution control elements. The task may trigger behavior by calling POUs, either on a periodic basis or upon the occurrence of the rising edge of a specified Boolean variable [4].

It is claimed that the main motivation for the IEC 61499 to adopt the event-driven paradigm is portability, which is defined as "the desire to make the code independent of the sequence of FB invocation in the PLC scan loop" [1, Section II-C]. Except that this definition is arbitrary, it is not clear how the event-driven paradigm and the "strong data encapsulation into components" [1, Section II-D] improve portability. It should be mentioned again that ISaGRAF does not implement the IEC 61499 based on the event-driven paradigm. The implementation of the event connections between FB instances using the event-driven paradigm is discussed in the next section. Moreover, in a similar way with the first feature, that is, processes, the author relates a design-time construct such as the event connection between FB instances, with implementation mechanisms such as the event handling mechanism of Java, the use of monitors or semaphores, or the interprocess communication mechanisms of operating systems. Most of the IEC 61499 run-time environments, including ISaGRAF and FBDK, do not implement the event connections using an event-handling mechanism.

5. Observations on Determinism and Event Driven

It is claimed in [1, Section II-C] that "FBs of IEC 61499 are event driven, that is, they remain idle unless an event is sent to one of their event inputs." The author also uses the term "event-driven execution" and "event mechanism" without defining their meaning. We start the discussion on this observation with some basic definitions.

Deterministic is a computation situation in which the execution time of an action sequence is known precisely [23]. It was realized that the traditional approach to achieve determinism, the one based on cyclic executives, that is, static scheduling, such as the scan-based model, is inflexible and difficult to maintain. This was the motivation for the scheduling theory in the domain of real-time systems [24]. The activation of the FBN of an application or subapplication can be either time triggered or event triggered. Both models are supported by the IEC 61131 through the periodic and triggered execution of tasks correspondingly. More specifically a task is defined, according to 61131, as "an execution control element which is capable of calling, either on a periodic basis or upon the occurrence of the rising edge of a specified Boolean variable, the execution of a set of program organization units, which can include programs and function blocks whose instances are specified in the declaration of programs." So, these models result in the periodic or triggered execution of the group of POUs that are associated with a task. The use of periodic tasks for the execution of POUs that capture the behavior of the application to an event or a set of events is also known as scan-based execution model. This model has been adopted in the FBN given in Figure 7 ([1, Figure 3]) using the E_CYCLE FB.

Unfortunately, the IEC 61499 standard does not define the semantics of the event connection between FBs. For example, it is not defined if the event connection represents a persistent or transient communication, an asynchronous or synchronous communication, not even if it is of type signal or invocation. This means that the following two models can be used, resulting in different behaviors for the same 61499 design specification.

(a) Event connections are implemented based on method calls that can implement both synchronous and asynchronous communications between FB instances. In the case of synchronous communication, the execution order is well defined during compile time. This is the model adopted by IEC 61131.

(b) Event connections are implemented using the event-driven paradigm that means that the order of execution of the behaviours captured by the FB instances of the FBN is unknown during compile time and is defined during run time. This results in an execution model that is more difficult to prove its determinism, compared to the one based on the synchronous method call.

Based on these definitions, we discuss in the following the "event-driven execution" and determinism as these are presented in [1]. Even though the event-driven nature of IEC 61499 FB is considered as one of the main features of the IEC 61499 [1, Section II-C], it is claimed that an implementation of an IEC 61499 design, based on the event-driven paradigm, may lead to nondeterministic behavior [1, Section III-A]. The "loss of events in case the queue capacity is exceeded" is reported as one reason for this. This is the reason, according to [1], for the appearance of several proposals for the execution of IEC 61499 based applications. The synchronous model of execution [2], the cyclic model of execution [25] (see [1, 25]), and the ISaGRAF model [20], which is considered close to the cyclic model, are given as examples of this kind of implementation, that is characterized as a cyclic execution model. Moreover, it is claimed that "these models "bend" a bit the fundamental concepts of event-driven invocation, implemented in the pure event-driven implementations, such as FBRT and FORTE." However, the cyclic execution model is not related with the event-driven paradigm since, as also stated in [25, Section II-B], this model "identifies the need to have a predefined order before execution, such that during the course of a scan, all FB's within a FBN will always be invoked in the same order."

Moreover, FBRT is not based on the event-driven paradigm, since it implements the event connections between

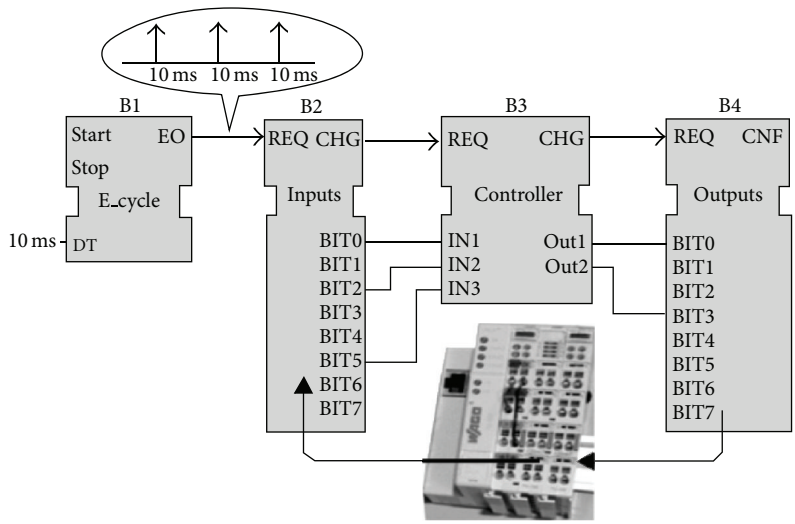

Figure 7: Function block application structure with periodic sampling of inputs [1].

FB instances, as method invocations, and this is why, as also claimed in [20], it cannot be regarded as a reference implementation. Example implementations based on the event-driven paradigm are described in [24, 26].

The proposal to sample external inputs periodically, making them available to the rest of the application in order to achieve determinism [1, Section III-A], has nothing to do with the way that event connections between FB instances are implemented. However, it is claimed in [1, Section III-A] that this model, that is, the scan based, "preserves the event-driven nature of the application (the block B3) and all associated benefits". The comparison of the proposal with the traditional PLC scan, based on Figure 7, as well as the arguments used in the corresponding discussion based on the fact that FB instances that do not have events will not be executed, bring no benefits regarding determinism. A detailed discussion on the event model of IEC 61499 and on the misperceptions and contradictions related with it is given in [27], where it is claimed that IEC 61499 fails in defining a coherent event-handling model.

6. Miscellaneous Observations

6.1. Observation on Design. ECC is not something new in industrial automation software. An analogous design notation, the sequential flow chart (SFC), has been adopted by IEC 61131 and can be used to define the behavior of programs or FBs. More specifically, as stated in [4], the "sequential function chart (SFC) elements can be used in conjunction" with either graphic or textual languages. Moreover, there is no means to define the behavior of composite FBs, since there is no ECC for the composite FB. It is claimed in [1], Section III E, that a similar effect with hierarchical state machines is achieved using composite function blocks. However, the composite function block is a construct to address structural complexity and allow the structural decomposition of FBs, while hierarchical state machines are used to address behavioral complexity and to apply abstraction in behavior modeling. A detailed discussion on the open issues in the execution semantics of ECC is given in [28].

6.2. Observation on Portability and Interoperability. It is claimed that portability and interoperability, even though "embedded" in the IEC 61499, have not been yet the major concern of vendors [1, Section IX]. However, no arguments are given on how portability and interoperability are supported by the standard except from the wish to achieve these features. The use of XML as a notation to specify the design documents provides a kind of portability for design-level artefacts. However, execution semantics should be well defined in order to achieve portability, and IEC 61499 fails to this direction [28]. As is also claimed in [29] "the standard is ambiguous and different implementations of the standard have made different assumptions to cope with the ambiguities." This has as result, the same application to behave differently when it is executed on different implementations, "thus making the applications unportable" [29].

Regarding interoperability, the only reference is to the research work presented in [30] ([1, Reference 31]) that describes a proposal to achieve interoperability using binary XML, which is also criticized to have impacts on interoperability, "as there are several different versions of binary XML, supported by different user groups" [1, Section III-C].

6.3. Observation on Performance. It is claimed in [1, Section III-C] that "XML tools are quite performance hungry and therefore not appropriate for many embedded platforms," and in order to address this drawback, the use of binary XML is mentioned. However, XML was adopted by IEC 61499 in a similar way with other modeling notations, IEC 61131 being among them, to get the benefit of portability of design time artefacts between development tools. It is clear that the notation used to represent the design-time models has no

effect on the performance of the executable model. Design specifications expressed in XML notation are transformed to executable models. This approach was adopted in [26] for which performance results are also given.

6.4. Observation on Intelligent Automation. It is claimed that the wider adoption of IEC61499 will help the industry to benefit from the promise of intelligent automation research results, but there is no reference to the constructs of the notation that supports the specification of intelligence. However, a system is characterized as intelligent according to Webster's Dictionary when it has or indicates "a high or satisfactory degree of intelligence and mental capacity." And intelligence is defined as "the ability to learn or understand or to deal with new or trying situations." The only argument used, that is, that the intelligence "is genuinely decentralized and embedded into software components, which can be freely distributed across networked devices," is also valid for IEC 61131.

7. Conclusion

The automation of an experimental shoe manufacturing factory and the tool that was used to obtain "revolutionary high level of manufacturing flexibility" are used as characteristic case study in [1] to prove that IEC 61499 is a solid technology for distributed and intelligent automation. However, it is known that the IEC 61499 implementation that is offered by this tool is based on the well proved and widely accepted IEC 61131 technology. Additionally, no arguments are presented to prove that an IEC 61499 implementation on an IEC 61131 one may exhibit considerable improvements regarding modularity, reusability, and reconfigurability compared with IEC 61131, as claimed in [1]. In this paper, several observations are also presented on the key features of IEC 61499, as these are presented in [1]. Determinism and the application of the event-driven paradigm are also discussed to identify misperception in the IEC 61499 function block model. This discussion raises many questions about the effectiveness of the IEC 61499 regarding the features mentioned in [1]. If we also take into account the claim of even its strong supporters, including the task force leader of the corresponding working group of IEC 61499, according to which "even stricter and more precise provisions are required in order to achieve the main goals of the IEC 61499 standard that are portability, (re)configurability, interoperability, and distribution" [6], it is evident that IEC 61499 has a long way in order to be seriously considered by the industry. Industry is looking for mature and stable technologies that will address new requirements, and IEC 61499 currently fails in several aspects to this direction, as it is claimed in this paper. Extensions and modifications are required to improve the standard. Then, the IEC 61499 community should prove that the new version of the standard provides a solid and mature infrastructure for the next generation of automation systems in order to be seriously considered by the industry. And this has to be done in the new era that is defined by the object-oriented FBD of the upcoming version of IEC 61131.

Abbreviations

DCS: Distributed control system
ECC: Execution control chart
FB: Function block
FBD: Function block diagram
FBDK: Function block development kit
FBN: Function block network
IEC: International electrotechnical commission
OO: Object oriented
PLC: Programmable logic controller
POU: Program organization unit
SFC: Sequential function chart
XML: Extensible markup language.

References

[1] V. Vyatkin, "IEC, 61499 as enabler of distributed and intelligent automation: state-of-the-art review," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 4, pp. 768–781, 2011.

[2] L. H. Yoong, P. S. Roop, V. Vyatkin, and Z. Salcic, "A Synchronous approach for IEC 61499 function block implementation," *IEEE Transactions on Computers*, vol. 58, no. 12, pp. 1599–1614, 2009.

[3] "The father of invention: Dick Morley looks back on the 40th anniversary of the PLC," Manufacturing Automation Magazine, http://www.automationmag.com/features/the-father-of-invention-dick-morley-looks-back-on-the-40th-anniversary-of-the-plc.html .

[4] International Electrotechnical Commission, "IEC international standard IEC, 61131-3: programmable controllers, part 3: programming languages," in *Proceedings of the International Epilepsy Congress (IEC '03)*, 2003.

[5] PLCopen, "PLCopen for efficiency in automation," http://www.plcopen.org/.

[6] T. Strasser, A. Zoitl, J. H. Christensen, and C. Sunder, "Design and execution issues in IEC 61499 distributed automation and control systems," *IEEE Transactions on Systems, Man and Cybernetics C*, vol. 41, no. 1, pp. 41–51, 2010.

[7] K. Thramboulidis, "Towards an object-oriented extension for IEC, 61131," in *Proceedings of the 17th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA '12)*, Krakow, Poland, September 2012.

[8] International Electrotechnical Commission, "International standard IEC 61499, function blocks, part 1-part 4," in *Proceedings of the International Epilepsy Congress (IEC '05)*, 2005.

[9] D. Witsch and B. Vogel-Heuser, "Close integration between UML and IEC 61131-3: new possibilities through object-oriented extensions," in *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation (ETFA '09)*, Palma de Mallorca, Spain, September 2009.

[10] D. N. Ramos-Hernandez, P. J. Fleming, and J. M. Bass, "A novel object-oriented environment for distributed process control systems," *Control Engineering Practice*, vol. 13, no. 2, pp. 213–230, 2005.

[11] K. Thramboulidis and G. Frey, "An MDD Process for IEC, 61131-based Industrial Automation Systems," in *Proceedings of the 16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA '11)*, Toulouse, France, September 2011.

[12] B. Werner, "Object-oriented extensions for IEC 61131-3," *IEEE Industrial Electronics Magazine*, vol. 3, no. 4, pp. 36–39, 2009.

[13] K. Thramboulidis and G. Frey, "Towards a model-driven IEC, 61131-based development process in industrial automation," *Journal of Software Engineering and Applications*, vol. 4, no. 4, pp. 217–226, 2011.

[14] L. H. Yoong, P. S. Roop, V. Vyatkin, and Z. Salcic, "A Synchronous approach for IEC 61499 function block implementation," *IEEE Transactions on Computers*, vol. 58, no. 12, pp. 1599–1614, 2009.

[15] G. Doukas and K. Thramboulidis, "A real-time-linux-based framework for model-driven engineering in control and automation," *IEEE Transactions on Industrial Electronics*, vol. 58, no. 3, pp. 914–924, 2011.

[16] D. Streitferdt, G. Wendt, P. Nenninger, A. Nyßen, and H. Lichter, "Model driven development challenges in the automation domain," in *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC '08)*, pp. 1372–1375, Turku, Finland, August 2008.

[17] S. Panjaitan and G. Frey, "Combination of UML modeling and the IEC 61499 function block concept for the development of distributed automation systems," in *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation (ETFA '06)*, pp. 766–773, Prague, Czech Republic, September 2006.

[18] K. Thramboulidis, D. Perdikis, and S. Kantas, "Model driven development of distributed control applications," *International Journal of Advanced Manufacturing Technology*, vol. 33, no. 3-4, pp. 233–242, 2007.

[19] A. Brusaferri, A. Ballarino, and E. Carpanzano, "Reconfigurable knowledge-based control solutions for responsive manufacturing systems," *Studies in Informatics and Control*, vol. 20, no. 1, pp. 31–42, 2011.

[20] V. Vyatkin and J. Chouinard, "On comparisons of the ISaGRAF implementation of IEC 61499 with FBDK and other implementations," in *Proceedings of the 6th IEEE International Conference on Industrial Informatics (IEEE INDIN '08)*, July 2008.

[21] D. Harel, "Statecharts in the making: a personal account," *Communications of the ACM*, vol. 52, no. 3, pp. 67–75, 2009.

[22] K. Thramboulidis, "Design alternatives in the IEC 61499 function block model," in *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation (ETFA '06)*, pp. 1309–1316, Prague, Czech Republic, September 2006.

[23] B. P. Douglas, *Real-Time Agility*, Addison Wesley, Pearson Education, 2009.

[24] T. Lui Sha, K. Abdelzaher, K. Arzen et al., "Real time scheduling theory: a historical perspective," *Real-Time Systems*, vol. 28, pp. 101–155, 2004.

[25] P. Tata and V. Vyatkin, "Proposing a novel IEC61499 runtime framework implementing the cyclic execution semantics," in *Proceedings of the 7th IEEE International Conference Industrial Informatics*, pp. 416–421, 2009.

[26] G. Doukas and K. Thramboulidis, "A real-time-linux-based framework for model-driven engineering in control and automation," *IEEE Transactions on Industrial Electronics*, vol. 58, no. 3, pp. 914–924, 2011.

[27] K. Thramboulidis, "IEC, 61499: back to the well proven practice of IEC, 61131?" in *Proceedings of the 17th EEE International Conference on Emerging Technologies and Factory Automation (ETFA '12)*, Krakow, Poland, September 2012.

[28] K. Thramboulidis, "IEC61499 function block model: facts and fallacies," *IEEE Industrial Electronics Magazine*, vol. 3, no. 4, pp. 7–26, 2009.

[29] G. Cengic and K. Akesson, "On formal analysis of IEC, 61499 applications, part A: modeling," *IEEE Transactions OnIndustrial Informatics*, vol. 6, pp. 136–144, 2010.

[30] K. H. Hall, R. J. Staron, and A. Zoitl, "Challenges to industry adoption of the IEC, 61499 standard on event-based function blocks," *Proceedings of the 5th IEEE International Conference on Industrial Informatics*, pp. 823–828, 2007.