Different Perspectives

The "Face to Face" department is a welcome new addition to the magazine where critical discussion of a topic is encouraged. In this department, two different viewpoints are offered regarding the International Electrotechnical Commission (IEC) 61499 standard. The first viewpoint (in blue) by Kleanthis Thramboulidis discusses the facts and fallacies of the IEC 61499 function block model. The second viewpoint (in orange) by Alois Zoitl and Valeriy Vyatkin summarizes the methodologies necessary for industrial automation in an IEC 61499 architecture.

IEC 61499 Function Block Model: Facts and Fallacies

by Kleanthis Thramboulidis

Control and automation systems in factory automation are developed using the procedural and device-centric paradigms. The always-increasing complexity of systems in this domain, as well as the need for agility, flexible plug-and-play, extensibility, and evolution, imposes the need for new paradigms to effectively address today's requirements. The function block (FB) model, introduced by the IEC 61499 standard [1], is an attempt to open the industrial systems market and exploit current software engineering practices and the application-centric paradigm in this domain. The IEC 61499 standard is also an attempt to address requirements such as interoperability, portability, distribution, agility, run-time reconfigurability, higher availability, and reliability. It is supposed to 1) facilitate the exchange of design information between designers and fabrication houses and 2) allow the designers to integrate competing vendors' tools and reduce the risk of relying on proprietary languages and data formats. However, even though the standard has been officially accepted by 2005, it is not yet adopted by the industry [2]–[4], and its status in the academic research community is questionable.

In this article, the current status of the standard is described, and its drafting process as well as its validation prior to implementation is commented. Facts and fallacies are presented and properly discussed to

Digital Object Identifier 10.1109/MIE.2009.934788

IEC 61499 Architecture for Distributed Automation: The "Glass Half Full" View

by Alois Zoitl and Valeriy Vyatkin

Control software is the main element in today's industrial automation system for providing correct and safe operation of the automation process. Furthermore, requirements such as flexibility, adaptability, or robustness, envisaged in the visionary study from the Iacocca Institute [1], largely increase the complexity of the control software to the extent where the existing design methods fail. Therefore, new means of developing the control software are necessary. With the new family of standards for distributed automation systems, called IEC 61499 [2]–[4], the IEC tried to proactively anticipate and fulfill these new and upcoming demands. The standard has been available since 2005 and has attracted substantial research attention, resulting in many publications (see surveys in [5] and [6]) and several reference implementations [7]. However, its industrial adoption is rather low. The main reasons for the slow adoption are unresolved semantic issues [8], lack of clear application and development guidelines, and missing industrial-grade implementation platforms [6].

The standard is hard to read and contains some ambiguities. Furthermore, only limited tutorial information on IEC 61499 is available. Just two books explaining the ideas of IEC 61499 were published to date: [9] and [10]. However, the first of these two is a little outdated as it considers the first draft version

Digital Object Identifier 10.1109/MIE.2009.934789

alleviate the confusion about the semantics of the IEC 61499 FB model, which is one of the most important reasons that the industry has not yet accepted the standard. The objective is to contribute to the direction of revealing some unsubstantiated claims that have been created around the IEC 61499 FB model and trigger a discussion in the community of this domain to critically ask itself "is the research and work we are doing going into the right direction?"

The IEC 61499 FB Model

The IEC 61499 FB was defined as an extension of the IEC 1131 FB to address today's challenges in industrial automation systems development. It is defined as a design level construct to encapsulate industrial algorithms and the data that these algorithms operate on. The FB type consists of a head and body, as shown in Figure 1, where the graphical representation of the PID_SIMPLE FB type is given. The head, which is used to capture the dynamics, accepts event inputs (EIs) and generates event outputs (EOs). The body, which is used to capture the functionality in terms of algorithms, accepts data inputs and generates data outputs. The FB is more than an object, as defined by the object technology, since it explicitly defines the way to capture its dynamic behavior. A specific kind of statechart, which is called execution control chart (ECC), is used to specify the dynamic behavior of its instances. Figure 2 presents the ECC of the PID_SIMPLE FB type. When in a state, the FB instance executes the associated EC actions. For each EC action, the algorithm is executed and its associated EC event is issued. The transitions that are leaving the current state are examined next.

An application is defined as a network of interconnected FB instances that accept inputs from the mechanical system through sensors and generate outputs that are sent to the mechanical system through actuators.

Even though many articles have been published on this subject over the last few years, the number of actual implementations, even prototypes, is very

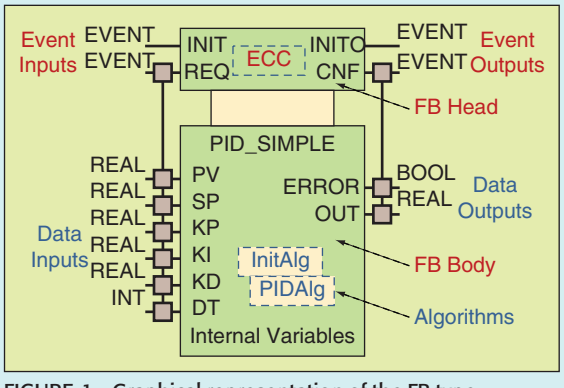

FIGURE 1 – Graphical representation of the FB type.

of the standard. This led to many misconceptions of the standard's ideas and contradicting conclusions of the researchers on the role and place of the standard in future industrial automation. For this reason, the majority of control device vendors and users (e.g., control engineers) still cannot appreciate the advantages of using IEC 61499.

This article attempts to bring some insight into the concepts and models of IEC 61499 by discussing and analyzing common misconceptions. We focus on two issues: modeling of distribution and architecture-centric design, which have not received substantial attention of researchers so far. Our analysis points out future research directions for increasing the adoption rate of IEC 61499.

Background and Current State of IEC 61499

For the last 20 years, the standardization and research efforts related to control software of industrial automation were focused on two main points: 1) improve the software quality and reliability and 2) reduce the development time of industrial automation control applications by reusing developed control software elements across different automation projects and also across control devices of different vendors. The latter issue is commonly referred to as vendor independence. The IEC 61499 was developed to meet these requirements. However, in this task, it has to compete with the well-established predecessor standard IEC 61131-3 [11], which has brought some relevant solutions or at least improved the situation of the users.

Unification and Modularization with IEC 61131-3

One of the main goals driving the IEC 61131-3 development was to unify the programming concepts of industrial control applications. At the time when the IEC 61131-3 development started, a great variety of different programming languages and concepts had been used in industry. The IEC reduced this variety down to five programming languages, some of which are textual and some graphical. Furthermore, it defined a common way of handling inputs and outputs, data types, and control programs. Currently, nearly every control vendor supports the standard, at least partially. Therefore, engineers knowing IEC 61131-3 can now switch easier between devices of different control system vendors. However, because of vendor-specific extensions or only partial support of IEC 61131-3, direct reuse of developed control software elements is not really possible.

For structuring control applications, the function block (FB) concept has been introduced. Similar to integrated circuits in electrical circuit design, an FB encapsulates a certain functionality and can be

limited. There is no mature reference implementation to demonstrate the applicability and advantages of the specification. Several prototypes or under-development integrated development environments (IDEs) exist to support the development process. The FB Development Kit (FBDK) (www.holobloc.com), the CORFU/Archimedes (http://seg.ece.upatras.gr), and the Framework for Distributed Automation and Control (4DIAC) (www.fordiac.org/) are currently the most well known in the community. There are also several prototype run-time environments such as function block run time (FBRT) (www.holobloc.com), Archimedes RTSJ-AXE [5], Archimedes RTAI-AXE [6], FORTE (http://sourceforge.net/projects/fordiac), and JAKOBI [7]. A small number of example applications have been developed to demonstrate the applicability of the specification [8], [9].

The Standardization Process of IEC 61499

The great influence of the IEC 1131 FB model [10] on the IEC 61499 standard can be easily identified. Even though the specification attempts to exploit current software engineering practices, it has many disadvantages regarding its theoretical basis in exploiting current software-engineering concepts and technologies, such as object orientation and component-based development. This is probably one of the most important reasons for the many ambiguities that exist in the specification. What Egyedi [11] argues with regard to the standards development process and specification, namely that "although the immediate problem usually lies in the way standards are implemented, the underlying causes are mostly flaws in the scope of standardization, in the standards process or in the specification itself," also applies to the IEC 61499 standard. Looking at the preparation stages for standards as defined from IEC [12], there is no concept-evaluation stage in

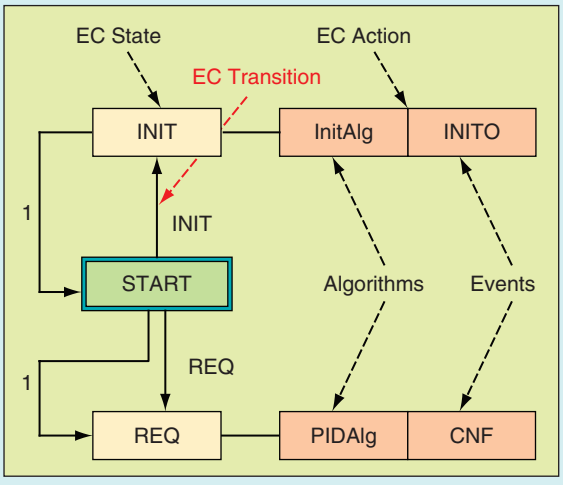

FIGURE 2 – The ECC is used to define the dynamics of the FB type's instances.

connected to other FBs via its data inputs and outputs [see Figure 1(a)]. In IEC 61131-3, an FB can contain just one algorithm, which may be written in any of the IEC 61131-3 programming languages. Because of this limitation, the FB concept of IEC 61131-3 can be seen as a procedural programming approach, where the FBs are the procedures, with FB inputs as parameters and their outputs as result. However, we must not neglect that an FB encapsulates not only one algorithm but also data, and that, it can maintain its state between invocations. These are the features of object-oriented programming languages that are more sophisticated in facilitating reuse. However, IEC 61131-3 does not support such object-oriented features as inheritance and interface concept.

The encapsulation of application parts in FBs helps to modularize control applications and foster the reuse of application parts. However, the IEC 61131-3 has two main drawbacks hindering the reuse. First, it allows global data. Global data acts as a hidden interface between the FBs and makes seemingly separated FBs tightly connected. This leads to rigid program structures, where parts may not behave the same without the remaining application part, and changes done locally in some application's parts may have unforeseen global consequences. The second drawback is that the application developer has no explicit control on the execution order of the FBs in an application. In an IEC 61331-3 control system, the execution order is derived from the connections between FBs, according to the rules defined in IEC 61131-3 [11, pp. 249ff.]. These rules leave some room

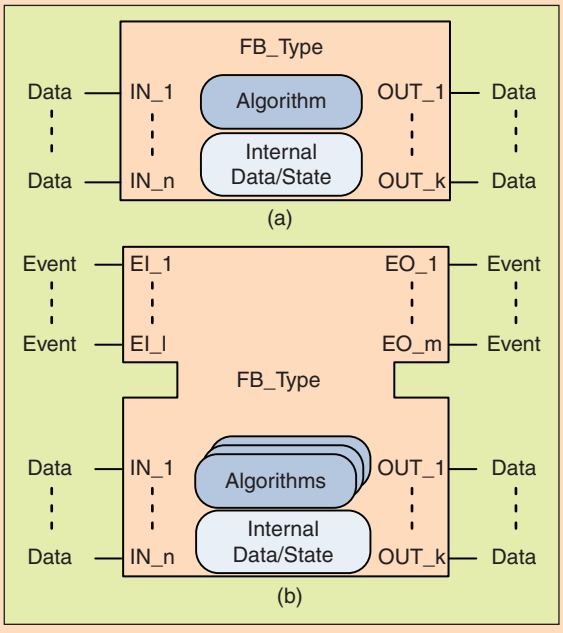

FIGURE 1 – The graphical representation of the FB concept in (a) IEC 61131-3 and (b) IEC 61499.

the form of a reference implementation before the acceptance of the standard. The FBRT, the first prototype implementation, could not be considered as a reference implementation by the time the standard was adopted, since it violates a lot of semantics defined by the specification. Moreover, other implementations were not utilized to revisit and resolve ambiguities in the standard.

Another problem lies in the way the standard defines the execution semantics of the FB model. IEC 61499 is an example of what is claimed in [11], namely that many standardizing organizations neglect standard implementation issues because this is argued to be a matter best left to the market. This is why the author in [11] recommends standard organizations to shift their emphasis from standard development to a more systematic inclusion of implementation concerns, both at the technical level of standard committees and at the policy level of standard organizations.

The Current Status and the Paradigm Shift

If we look at the current status regarding the adoption of IEC 61499 FB model, we can see both a promising and a disappointing view. The academic view seems to be the promising view, while the disappointing view is the industry's view.

Many researchers from universities and research institutes are working to exploit IEC 61499 in industrial automation. Even though this research has resulted in several prototypes or under-development IDEs and run-time environments, the number and complexity of the example applications presented to demonstrate the applicability of the specification are not sufficient to demonstrate the maturity of the new technology. The great number of publications is not definitely an advantage, since many contradicting statements on the FB model impede a clear understanding of the technology and its advantages. It is difficult even for members of the IEC 61499 community to discriminate between facts and fallacies, between facts and myths.

On the other hand, the industry's view is a disappointing view. Even though the standard was officially accepted in 2005, there is no indication that the industry is going to adopt the standard. Only recently, ICS Triplex (http://www.icstriplex.com) announced that ISaGRAF provides support for the IEC 61499 FB model. An impressive demonstration has been established and presented at the 12th IEEE Conference on Emerging Technologies and Factory Automation (ETFA) 2007. Two applications, a train simulation and an orchestra, were used to demonstrate the features of the new technology [13].

From our experience over the last few years in this domain, we can state that there is a tendency for interpretation; therefore, the same application may work differently on different control devices.

Component-Oriented Design with IEC 61499

The mentioned weaknesses of IEC 61131-3 partially stem from the fact that the standard is now more than 15 years old, so its concepts are not the state of the art in software engineering anymore. However, new software engineering trends cannot be taken directly into the industrial control systems domain. There are many specific requirements of the automation developers, and a typical control engineer has only limited knowledge in computer science and software development. The IEC took this into account for the development of the IEC 61499 architecture, which should support such new features of next-generation industrial automation systems distribution and reconfiguration [1].

To leverage the existing know-how, the IEC 61499 architecture builds on top of the IEC 61131-3 definitions. The main element of the architecture is the FB, but this concept has been extended in several ways to incorporate new developments from the domain of software engineering [Figure 1(b)]. The most eye-catching extension is the event interface. An FB in the IEC 61499 remains passive until triggered by an input event. On event, the FB executes and produces output events and data. On the one hand, the event interface complicates the design when compared with IEC 61131, adding new connections between FBs. However, the event interface in IEC 61499 allows for explicit specification of the FBs' execution sequence. This gives the developer a new level of flexibility not possible in IEC 61131-3.

Another potential problem created by event-triggered execution of the FBs in IEC 61499 is the specification and implementation of behavior under real-time constraints. However, recent research results also overcome this limitation (see [12] for an overview on execution methods for IEC 61499 applications complying with real-time constraints).

There are a few other extensions of the FB concept in IEC 61499 toward object orientation and component orientation. First, the FBs may contain several algorithms similar to the methods encapsulated by an object. However, in contrast to an object, the algorithms inside an FB are neither visible nor directly accessible from the outside. Furthermore, there is no global data in IEC 61499. This greatly enhances the reusability of FBs as there are no implicit dependencies between application parts; therefore, removal or addition of an FB influences only the connected FBs. Internal variables of an FB are also completely hidden. There are no means to access or change an FB's internal variable

for the industry to reject the IEC 61499 standard simply because 1) there is no mature reference implementation to demonstrate the applicability and the advantages of the new technology and 2) its learning curve is perceived as being very steep.

As far as the second reason is concerned, the most important challenge is to allow the industrial engineer to easily change the way of thinking during the system's development process. This change of thinking, known as a paradigm shift, is required because industrial engineers are familiar with 1) the device-centric and procedural-based paradigms that are adopted by current practices in industrial systems development and 2) the IEC 1131 standard that is widely used in the industry.

It is clear that the FB model is not only a new technology in the domain. It promotes the application-centric approach and partially adopts the object-oriented (OO) one. It is not just a packaging mechanism for procedural programming, and this is why a change in mental model in the practitioners, i.e., a paradigm shift, is required [14]. This means that a specific strategy should be defined to make this paradigm shift easier for industrial engineers. This paradigm shift is more difficult than the one confronted by the software community regarding the transition from the procedural to the OO paradigm [15], because it should also be accompanied by a shift from the device-centric paradigm to the application-centric one.

Facts and Fallacies

A great number of articles on the IEC 61499 FB model with many contradicting assumptions and statements make the adoption of the standard a difficult task. In this section, we present the facts and fallacies in this domain and discuss open issues. We discuss incorrect specifications of the standard as well as unsubstantiated claims on the way that the IEC 61499 FB model has to be implemented. The main objective is to trigger a major revision of the standard that is greatly required for it to be seriously considered by the industry. The main challenges discussed in this section are 1) requirements elicitation; 2) programming in the large; 3) location transparency in design space; and 4) run-time environments.

Requirements Elicitation

Unsubstantiated Claim 1: The FB Network Can Be the First Application Model in the Development Process

The standard defines the FB type as a basic design-level construct; it does not refer to requirements specification and its evolution to design models. Moreover, it is widely considered in the IEC 61499 community that the FB network (FBN) can be the first specification of the application in the development process. as it was possible in IEC 61131-3 with the access path mechanism.

Taking these properties into account, an FB in IEC 61499 is an independent software entity that can be implemented, tested, and used independently of other FBs. Therefore, IEC 61499 much better supports the development and reuse of tested components (i.e., FBs), which will lead to a better quality of industrial automation software.

Current Industrial Adoption of IEC 61499

IEC 61499 potentially brings many benefits for developing industrial automation systems. These were proven in numerous case studies conducted in academia and research institutes worldwide (see the review in [6] and [7]). However, the current adoption of IEC 61499 in the industry is still very limited. One of the first industrial automation engineering tools supporting IEC 61499 was ISaGRAF [13], a product of ICS Triplex (currently, a part of Rockwell Automation). Since Version 5 was released in 2005, this IEC 61131-3 product has been enhanced with the support of many of the IEC 61499 elements. It is now possible to develop distributed control applications in IEC 61499 together with the application parts in IEC 61131-3. However, not all concepts of IEC 61499 have been implemented. Thus, the communication between the application parts is achieved through network variables instead of service interface FBs (SIFB). The appearance of FBs in ISaGRAF follows IEC 61499, but their internals look a bit differently. For example, the execution control chart of a basic FB is defined by means of the IEC 61131-3 sequential function chart language. The event-driven IEC 61499 FBs are executed on top of a cyclic-scanned and time-triggered IEC 61131-3 run-time system. Despite the mentioned limitations, capabilities of ISaGRAF have been demonstrated in distributed systems consisting of up to 70 controllers [14]. Several pilot systems in research organizations have been implemented using the IEC 61499 feature of ISaGRAF, but no real industrial deployment was reported so far to the best of the authors' knowledge.

Another example of an industrial-scale IEC 61499 development is the Austrian company nxtControl that has developed an integrated supervisory control and data acquisition and distributed control approach based on IEC 61499 [15]. They provide an industrial-grade engineering environment that supports the design of control applications and visualization together in one tool. This approach has great advantages in productivity and reuse of both control and visualization components. The nxtControl engineering tool provides several features that have been long expected from IEC 61499, for example, debugging and online-monitoring infrastructure, allowing to

According to this, the development process is considered as an integration process of already-existing FB types. Even though already-existing FB types can be used as a starting point, a lot of other FB types are required to capture the specific application's logic, and the control engineer has no guidance to this direction. The hybrid approach [16] that integrates the Unified Modeling Language (UML) notation with the FB model proposes a solution to this problem. However, more work has to be done for the better exploitation of currently used industry diagrams for the specification of the mechanical part of the plant [17]. Examples of these diagrams are the process and instrumentation diagrams (PI&Ds) and the procedure function chart (PFC) of the International Society of Automation (ISA) SP88. The IEC 62424 [18] describes how process-control engineering requirements such as requirements for process control equipment and functionality required by the control application may be captured by the P&ID. This extended P&ID may be used as a source of requirements to help the control engineer in the construction of the FB design model. It may also be used in safety analysis, and in general, it may be used as the basis for the coordination of the different disciplines involved in the development process.

Programming in the Large

Unsubstantiated Claim 2: The FB Is a Component

The size and complexity of the embedded software in industrial systems are increasing rapidly. At the same time, more quality properties such as portability, extensibility, predictability, flexibility, reliability, and security have to be guaranteed by these applications. Developing systems that achieve these characteristics is hard following the programming in the small paradigm. As the size and complexity increases, the design and specification of the overall system structure becomes more significant than the choice of algorithms and data structures. By concentrating on the structure of industrial system, it will be possible to overcome shortcomings in current engineering practices. A well-organized structure of the industrial system will reduce the complexity; it will provide a better understanding of how the system works, and it will also allow the analysis of the system's properties without implementing it. A solid software architecture will reduce the complexity and will thus lead to a better understanding of how the system works. So, the following questions need to be answered:

1) Can the IEC 61499 be considered as a notation to define the architecture of the system?
2) Can the IEC 61499 be considered as a higher level of abstraction compared with the one provided by the IEC 1131 FB model?

debug single FBs as well as fully distributed applications. Another feature is the automatic generation of the communication during the distribution process of the application. This greatly reduces the engineering effort when distributed control applications are designed. nxtControl has already applied its technology to eight projects in the domain of building automation. The largest project has been a training center building with 19 control devices, controlling about 2,500 inputs and outputs (I/Os) (such as heating, ventilation, air conditioning, and lighting) with IEC 61499.

Several other companies are currently investigating how and when to move to or integrate IEC 61499 in their product lines. One reason for the moderate adoption pace is the switching effort, which when compared with the advantages of IEC 61499 over the established IEC 61131-3 may seem to be only marginal. There is a lack of comprehensive case studies comparing the overall benefits and drawbacks of both technologies.

Possibly, IEC 61499 may be beneficial only in some domains of industrial automation applications, and this requires further investigation and case studies. However, we see some exciting features of IEC 61499 that will help to achieve higher quality control applications and therefore deserve a closer look. In the following, we will point out some of the features that have not got the attention they deserve.

Platform-Independent Application Design

The design of distributed control systems is obviously more complex than of the traditional centralized ones. To cope with that complexity, IEC 61499 offers a modern platform-independent approach to system design, similar to the model-driven architecture (MDA) [16] used in the development of complex software and embedded systems. MDA consists of three main models allowing to develop applications in a generic model-driven way. These models are the platform-independent model (PIM) for modeling the application; the platform-definition model (PDM) for modeling the target system (i.e., devices and the communication infrastructure); and the platform-specific model (PSM) that contains the assignment of the PIM elements to the devices and platform-specific configurations and adaptations. The PSM is used to automatically generate the target-specific code that will be executed in the devices. The MDA approach has greatly improved the flexibility and efficiency of the development process of embedded systems [17] on account of reusing elements of the solutions, described in high-level languages. The same solution can be easily implemented on a variety of targets, ranging from the code running on a standard hardware to fully customized hardware, implementing the same function.

There is currently a trend in the IEC 61499 community to consider the FB type as a component. For example, in [19], the authors referring to the component model of IEC 61499 claim that the FB possesses the required characteristics of a software component. In [20] and [21], the authors consider the IEC 61499 as a high-level architecture. In [21], the FB type is characterized as a component with event and data interfaces. It is also stated that the concept is analogous to the ideas of component, such as software component from software engineering. In [22], the FB is characterized as a high-level concept and also as a component with a state machine inside. The authors base this statement on the argument that the FB type is not relying on a particular programming language, operating systems, etc. However, by adopting this argument, we can also conclude that the process construct of the well-known data flow diagram (DFD) notation is a component, since it is not relying on a particular programming language and operating system.

To better understand the semantics of FB notation, a mapping of the FB type concept to the well known in real-time system development DFD notation is given in Figure 3. From this mapping, it is clear that the interface definition of the new construct is too low level at least as far as the data representation is concerned. The concept of data flow is not adopted in the FB model. Instead, a detailed representation of the constituent parts of the data flow is represented, cluttering the design diagram with too much detail. A detailed discussion on this mapping can be found in [23].

If we consider the current use of FB construct, we can see that it is mainly used, as for example in [21] and [24], to represent simple processes or functions. This approach is also adopted for the construction of the basic library of FBDK. The example given in [21] that uses the FB to represent the operation that calculates the expression $X2 - Y2$ is misleading for the objectives of the FB model, since it uses a construct defined for object representation to represent a

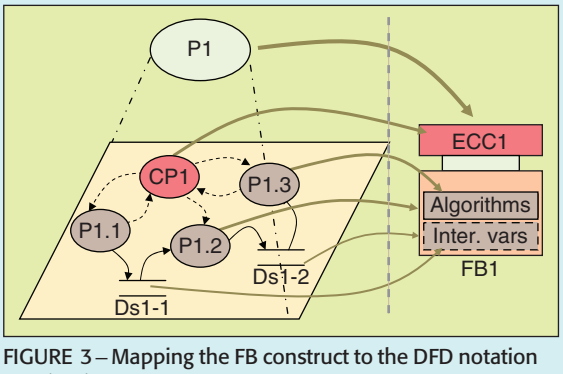

FIGURE 3 – Mapping the FB construct to the DFD notation constructs.

The design process promoted by IEC 61499 is subdivided into two steps. In the first step, the functionality of the whole system is defined using a PIM called application model [2, pp. 21ff]. This may require a detailed specification of each function as a composition of simpler functions. The application model is fully executable; however, it is free from particulars of hardware and communication protocols.

In the second step, details of a particular hardware configuration are taken into account in the form of a PDM called system model [2, pp. 18f]. The functionality is said to be mapped to that hardware, resulting in a PSM called distribution model [2, p. 26].

We can expect similar benefits from IEC 61499 for industrial automation that MDA brought to software engineering and embedded system development. In the following subsections, we will illustrate on a simple example the application-development process of IEC 61499.

The Application Model

In IEC 61499, the PIM of control applications is done by instantiating and interconnecting FBs. A simplified example of such a control application is shown in Figure 2(b). The application implements a closed-loop control. The FB SENSOR provides sensor data, and this data is taken by the FB control, which performs the control algorithm, whose output is passed to the actuator represented by an FB actuator.

This example shows a further difference to IEC 61131-3: I/Os are not directly addressed in the application. Instead, interface to the I/Os is implemented in the form of FBs. This helps to isolate dependencies on particular I/O names and addresses, keeping the core function free of such dependencies.

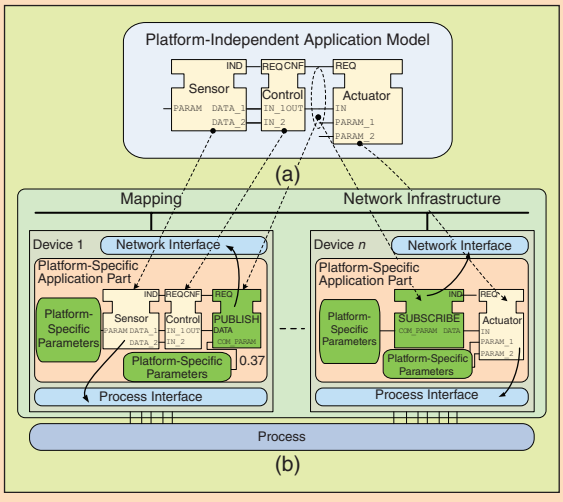

FIGURE 2 – Overview on the (a) platform-independent application model of IEC 61499 and (b) its relation to the control devices in a distributed control system.

function. This example can also be used as a representative of the level of detail that the IEC 61499 introduces in specification. This is clear if we compare this FB or the one that encapsulates more than one operations on X and Y, which is consistent with the OO approach, with the method signature of the operation that implements the calculation OUT $= X2 - Y2$, and the method $X2$minus$Y2()$ of one of the provided interfaces of the corresponding UML component, as shown in Figure 4. It should be emphasized that both the EIs and EOs and data of the $X2Y2$_ST FB type are the equivalent of the specification of just one method of a provided interface of a component. Moreover, the FBs that implement operations, as mentioned earlier, cannot be considered as components in a component-based development. It should be noted that it is not the graphical representation that makes a design-level construct component but its properties. This is why the "process" construct of the DFD notation is not a component.

The interface of a component has to define the operations provided. Instead, the interface definition of FB types is usually defined in a way that does not include any semantic information. For example, the graphical interface of the $X2Y2$_ST FB type, with input-event REQ, data inputs X and Y, output-event CNF, and output-data OUT, provides no information on the operation represented. This is also valid for the FB that encapsulates more than one operation on X and Y. Moreover, the name REQ that is used to identify the algorithm has no information on the

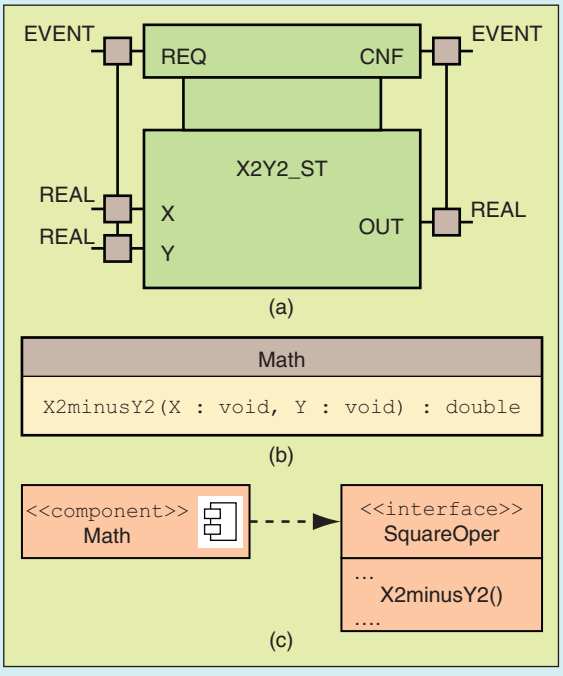

FIGURE 4 – IEC 61499 FB type versus class and component. The (a) FB, (b) class, and (c) component.

The direct access to the readings of peripheral controller I/O modules is implemented by a special type of FB called SIFBs [2, pp. 43–54]. A library of SIFBs can be specific for a particular device hardware, but there are many possibilities to organize I/O access in a generic way, for example, using a generic input FB, with parameters specifying type and address. As an option, such a generic FB can also implement simulated I/Os that are set/observed from screen. In early design stages, an application can be developed and tested using such generic I/Os.

The SIFB concept of IEC 61499 has often been criticized for its platform dependency. In our view, such a critique is based on the incorrect assumption of using SIFBs with fixed addresses of I/Os at the early design stages.

The System Model

To execute an application on a distributed network of control devices, it is necessary to take into account the particulars of the platforms composing the system. In IEC 61499, this can be done in the system model. Properties of control devices are described in IEC 61499 by the device model [2, pp. 19f]. They contain a network interface for communicating with other devices, a process interface for interacting with the controlled process, and a library of FBs provided in the device (e.g., special SIFBs for I/O access). A device can contain resources [2, pp. 20f] that provide an execution environment for applications. Resources can be regarded as independent execution containers for FB applications. They may provide specific functionality (e.g., special computation hardware), but may also be used just for subdivision of the device into smaller independent logical entities.

The communication infrastructure of a particular system is modeled with so-called segments and links [2, pp. 18f]. A segment describes a network segment of a certain type [e.g., control area network (CAN)] to which several devices are connected with links. One device can be connected to more than one segment (e.g., bridging or routing devices). This allows to freely model the hierarchical structure of the network topologies typically used in industrial automation.

In our example from Figure 2, we use just two devices: one connected to the sensor and the other to the actuator of our control loop. We also have just one network segment connecting the two devices with each other.

An open issue of IEC 61499's system model is that the descriptions of devices and segments are very generic. No specific parameters or methods are provided to describe the needed communication or device parameters. Communication parameters would be needed, for example, to allow the tools for checking the available

specific algorithm. The example given in [21] that implements $X2 - Y2$ as a network of FBs further confirms our arguments.

The FB type is also used by ISaGRAF, the first commercial implementation of IEC 61499, to represent a process that was traditionally represented by an IEC 1131 FB. However, for a commercial tool, this can be considered as an intermediate step to simplify for the industry the paradigm shift from the procedural IEC 1131 to the object-based IEC 61499 paradigm. The shift from the purely procedural IEC 1131 to the object-based IEC 61499 will be very cumbersome if such an intermediate level is not provided, since industrial engineers are not familiar with the concepts of object and component. This can be compared to the paradigm shift from the procedural to the OO paradigm in the software domain, where C++ allowed an intermediate-level shift. As a first step, many programmers used the C++ language to program in a procedural way following a smooth shift from the procedural paradigm to the OO one.

According to Szyperski [25], [26], a software component has to be, among other characteristics, a unit of deployment and thus has to be an executable deliverable for a (virtual) machine, so no human intervention will be required for its use. It also has to be a unit of versioning and replacement, and thus, it should remain invariant in the place of installation as it gets installed onto possibly many systems. According to [27], the design issues in the architecture level involve an overall association of system capability with components, where components are modules, and interconnections among modules are handled in a variety of ways. At the architecture level, the components are programs, modules, or systems. According to [28], components in UML 1.4 have interfaces, they are used to capture in diagrams the overall topology of the application, and they are mapped to hardware as architectural units. In UML 2.0, components extend classes with additional features such as 1) the ability to own more types of elements than classes can; for example, packages, constraints, use cases, and artifacts and 2) deployment specifications that define the execution parameters of a component deployed to a node [28]. According to [29], a component defines its behavior in terms of provided and required interfaces.

Taking into account the current use of FB and the aforementioned definitions of component, it is clear that the FB cannot be considered as a component for programming in the large. It is a construct that supports programming in the small, and thus FB libraries, as the one defined in FBDK, favor reuse in the small. Another construct is required for the FB model to be able to effectively support programming and used bandwidth. For device types, the main shortcoming is in the description of the communication and process interface. There is an extension toward specific ports that would be needed to know which networks a device supports or which I/O a device provides.

The Distribution Model

The final phase of the application-development process in IEC 61499 is the distribution of control application to the control devices. In this step, the application's FBs will be mapped to the control devices where they will be executed on. In our example, the SENSOR and the CONTROL FB are mapped to Device 1, and the ACTUATOR FB is mapped to Device n. This mapping is not just a logical link in the models; instead, a copy of the mapped application parts is created. The reason for this is that device-specific changes may have to be done. Such changes are the specification of device-specific parameters to the FBs. In our example, we have to define the device-specific parameters for the SENSOR and the ACTUATOR FB so that they know which I/Os they have to provide.

As a result of distribution, some FBs, connected to each other in the application via event and data connections, may reside in different devices. Such connections, going across device boundaries, need to be implemented by inserting communication SIFBs [2, pp. 47f]. In our example, these are the PUBLISH and SUBSCRIBE FBs added in the device-specific applications. In more complex applications distributed across many devices, the generation of communication FBs can be rather burdening. However, provided with the information from the application and system model, the design tool could automatically insert these FBs and provide suggestions for suitable communication parameters. The nxtControl Studio tool is the first to provide an implementation of this feature, which proved its efficiency in the development of highly distributed systems. ISaGRAF inserts communication functions implicitly, making them hidden from the user.

In our sample application, the mapping was done by copying the application parts to the device representations forming the PSMs together. For simplicity, we assumed that the generic SIFBs used in the application model are provided by the control device. In a general case, the application will just specify with a generic SIFB that it needs, say, a digital input. When this generic wish is mapped to the device, the specific SIFB providing the digital input on this device has to be inserted manually or by a tool.

Architecture-Centric Development with IEC 61499

Architecture-centric development is currently the dominating approach in the design of complex

in the large and allow the definition of system's architecture in such a way as to influence its efficiency, adaptability, and reusability of its software components. Such an extension should allow the definition of an abstraction of the system under development to suppress all these details of its elements that do not affect how they use, are used by, relate to, or interact with other elements of the system. It should allow the definition of the structures of the system, which should comprise 1) the software elements of the system (components), 2) the externally visible properties of those elements (interfaces), and 3) the relationships among them (ports and connectors). The UML component diagram [30] can be used to address programming in the large. In this case, the way to realize UML component diagrams with FB design diagrams should be defined. Architectural description languages may also be used.

Location Transparency

Unsubstantiated Claim 3: The IEC 61499 FB Model Is a Platform-Independent Model

The advantages of the platform-independent model (PIM) in the system's development process are well known [31]. A PIM should describe the software system, which supports a part or the whole of the industrial process system, in a highly abstract way that is independent of any implementation technology. It should model the system from the perspective of how it best supports the industrial process, without taking into account the configuration and type of technology on which the application will be implemented. The platform-specific model (PSM) will later describe in detail how the PIM will be implemented on a specific platform or technology.

The IEC 61499 introduces the service interface FB (SIFB) to be used in the FBN to implement event and data transfer over the network. The use of SIFB in the design diagram complicates the FBN diagram, completely destroys location transparency, and makes it a PSM. The developer has to work on this complicated, execution environment and configuration-specific FBN for any change, even in the case of a very simple one that concerns the introduction of a new FB instance or the reassignment of an FB instance to a new device that can be imposed by a change in the network of devices.

To exploit the benefits of PIM, the design diagram should be defined for a target technology-neutral virtual machine. Such a virtual machine that is defined as a set of parts and services, which should be defined independent of any specific platform, is shown in Figure 5 and is called IEC 61499 virtual bus.

software systems. It improves quality and reusability of the product. The software architecture should be defined at early development stages based on the requirements to the system.

So far, IEC 61499 has been commonly understood as a programming language for implementation of control algorithms. This view is limiting and not sufficient for designing complex industrial automation control systems. By definition, IEC 61499 defines a reference architecture. Therefore, it has several means to capture architecture, application structure, and requirements already at early stages of the application-development process. These means however have not attracted proper attention of researchers so far. In the following subsections, we will discuss some of them.

Application Structuring with Subapplications

A typical top-down application-development process starts with specifying the top-level application components and their interaction. In subsequent design steps, these components are specified in detail. The FB concept is only partly suited for such a top-down application development since the FB is atomic. That means that an FB can be later assigned only to one device. However, top-level or even medium-level application components may encapsulate the functions of several control devices (e.g., the control of a whole machine). Therefore, other encapsulation artifacts are necessary, which allow the internals of such components to be distributed to different devices. IEC 61499 provides such a design artifact called subapplication [2, pp. 37–39].

The top-down application development does not cover all the design scenarios, and the bottom-up approach is also necessary. In the bottom-up approach, the application parts can be grouped to subapplications. This can be illustrated on our closed-loop control example. We can encapsulate it in a subapplication providing the interface of the closed-loop control to higher levels (e.g., updating the set point) while still being able to distribute the FBs to the two control devices (Figure 3).

Currently, only the open-source engineering tool 4DIAC-IDE [18] provides some basic support of subapplications, which allows to group application parts together. To be fully usable, further extensions toward saving subapplications or loading subapplication templates as suggested by [19] will be necessary.

Typed Interfaces with Adapters

Another problem of IEC 61499-based design is clarity and readability of applications composed of many large FBs. Such FB diagrams usually have many data

It should include parts, such as a FB-type container and FB-instance container, and services that provide the functionality of SIFBs, management FBs, and event FBs. This virtual machine (bus) will be next realized in platform-specific ways on different execution platforms. Two prototypes that implement this proposal have already been described [5], [6]; published performance measurements show that both satisfy stringent real-time constraints. This approach also fulfills the objective of the standard, which is to support run-time reconfiguration. However, most of the existing or under evolution run-time environments adopt the monolithic application approach.

Run-Time Environments

Unsubstantiated Claim 4: A Profile Will Define the Execution Semantics of IEC 61499 FB Model

The term "profile" is used in standardization to define an agreed-upon subset and an interpretation of a specification. It is used in complex technical specifications that have many optional features so as to define different conforming implementations that may not interoperate because of choosing different sets of optional features to support. This is also the meaning of the term in UML. A profile not only does it conform to the semantics of general UML but also specifies additional constraints on selected general concepts to capture domain-specific forms and abstractions [32]. According to the earlier definition of the term, it is clear that there is no need for a profile to define the execution semantics, but the standard has to provide a clear definition of the underlying IEC 61499 metamodel at least as far as the execution semantics are concerned.

An IEC 61499-compliant run-time environment should provide a platform-specific implementation of the IEC 61499 virtual bus, as shown in Figure 5. This means that the run-time environment should not affect the design of FB types and network

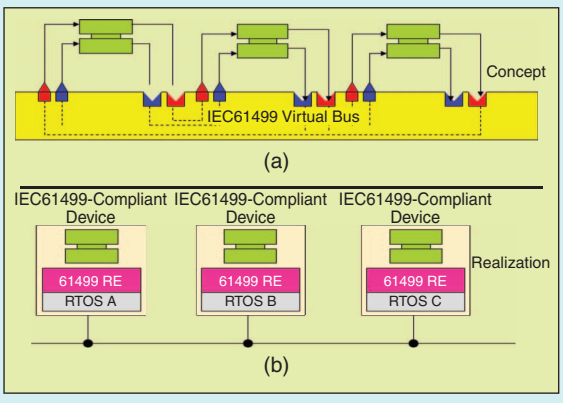

FIGURE 5 – The IEC 61499 virtual bus (a) and its realization (b).

and event connections cluttering the design space and making hard the understanding of FBs' interaction.

However, IEC 61499 provides a solution based on the adapter concept [2, pp. 39–43]. This concept allows grouping together events and data to form an interface that is represented as an FB. It is similar to the concept of ports used in UML 2.0. Ports simplify the interface of components by grouping of interface elements that logically belong together.

There are two different ways of using an adapter in FB interface definition: an interface accepting adapter called plug or an interface-providing adapter called socket. In the interface definition of an FB, the plugs are associated with the input side, and sockets are associated with the output side. The definition of an adapter type is done from the plug perspective, because the plug side of an adapter connection is typically the requirements-defining side. The socket has always the mirrored interface of the plug. This means that the events and data going into the plug are coming out of the socket and vice versa for the plug's outputs. Apart from reducing the number of connections, the adapter concept has the great advantage that it increases the decoupling of application parts. The plug or socket user needs no knowledge on the other part that they will be connected to.

We illustrate the use of the adapter concept in our example from Figure 2. Let us assume that our control loop is a pressure control loop. In this case, our control algorithm would need the process value delivered from a pressure sensor. To make the control algorithm independent from a particular pressure sensor type, we define a pressure sensor adapter. Our control algorithm will initialize the sensor giving it a pressure range it should deliver.

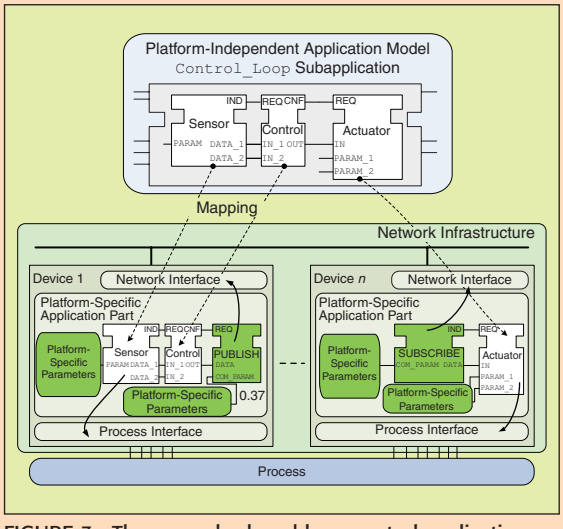

FIGURE 3 – The example closed-loop control application encapsulated in a subapplication.

diagrams, i.e., there is no need for the designer to know the specific characteristics of each run-time environment. The whole design should be based on the semantics of the virtual machine. This means that the execution semantics of an FB instance and network should have been clearly defined by a standard in a platform-independent way. It should have been defined in such a way that the design models be detailed and precise enough to produce fully executable models. It should also permit the automatic generation of the implementation models in the form of efficient code that can be executed on specific run-time environments. The FB design models should be behaviorally expressive and rigorous as well as intuitive and well structured. Unfortunately, the standard fails to rigorously define the semantics, and this makes the development of run-time environments a very hard task. This is also a major source of incompatibility between the different platforms as far as the execution of applications is concerned.

Since the IEC 61499 standard is intended to be used by control engineers in specifying distributed control systems, clarity and simplicity should be considered as key parameters for deciding upon execution semantics. Control engineers should be able to understand how their models work in a relatively intuitive way. So the semantics have to be rich enough to support different styles of modeling and at the same time be simple and intuitive.

There are several proposals regarding the execution of FB instance and network. Some of these have already provided a reference implementation; others are just specs without reference implementations or a theoretical proof of concept. This makes the development of run-time environments a very hard task. This subsection discusses these proposals, with the objective to identify the pros and cons that can be used to complement the standard in the direction of execution semantics.

The standard introduces many sources of confusion regarding execution semantics. One example is the so-called ECC operation state machine that is supposed to define the dynamics of ECC. Actually, this state machine defines the dynamics of the FB instance; the behavior of the FB instance is defined by the ECC of the corresponding FB type. The ECC operation state machine is the main source of confusion regarding the FB instance execution. The statechart shown in Figure 6 was constructed to clearly describe the dynamics of an FB instance, as close as possible to the one described by the standard, for the case that the FB instance is implemented as active. An FB instance may be defined at PIM or PSM as active or passive. An active FB has its own thread

Furthermore, we will request it for a new pressure value. The sensor should deliver the pressure in pascals and signal if there is any problem. The resulting adapter (from the plug view) can be seen in Figure 4(a). The usage of this adapter in the `Pressure_Control` FB and in the `Pressure_Sensor_TypeA` FB is shown in Figure 4(b). As the adapter has been defined according to the needs of the pressure control algorithm, the plug can be directly used in the `Pressure_Control` FB. In `Pressure_Sensor_TypeA`, there are some adjustments necessary. In our example, the pressure sensor delivers its values not within the correct range. Therefore, the FB `RANGE_ADAPT` transforms the sensor value to the correct range. Note the mirrored interface of the socket in the `Pressure_Sensor_TypeA` FB.

Describing Interface Semantic

The interface of an FB as defined in IEC 61499 is an important feature for designing safe and reusable software components. However, according to [20, pp. 50ff], the interface specification alone is not sufficient to ensure the use of a software component as a black box in different applications. Also, the interface semantic have to be defined. The interface semantic describe how the interface has to be used (e.g., first, event A and then event B), or how it behaves. For specifying the behavior of adapters and SIFBs, IEC 61499-1 prescribes in Clause 6.1.3 the use of International Organization for Standardization (ISO)/IEC 10731 time-sequence diagrams. According to this description, the sequence of input and output events and the values of data input and outputs can be defined with time-sequence diagrams. Although the

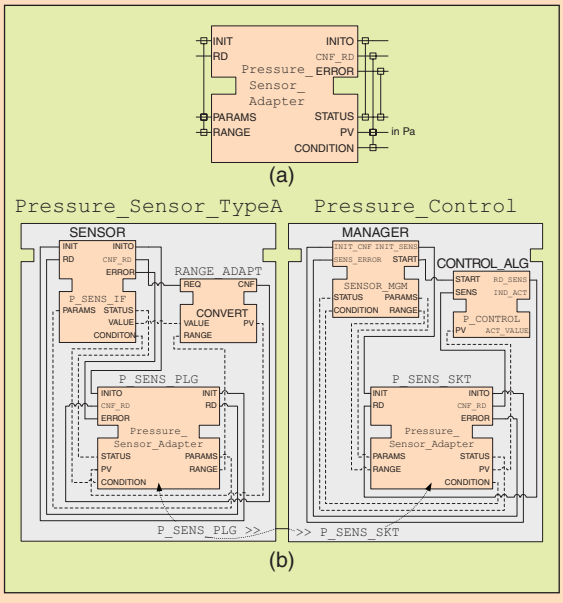

FIGURE 4 – (a) An adapter FB for a pressure sensor and (b) its usage in a pressure control application.

of execution, while a passive FB does not have its own thread of execution; it is executed by other threads. It is clear that the assumed design-level behavior for the FB instance should be the same in both cases. The same behavior should also be ensured for the case of an event-based implementation or a cycle-based one.

The execution semantics of the FB instance that are considered so far as undefined by the standard can be classified into three categories: 1) queuing or losing of input events; 2) scheduling function; and 3) event-processing policy.

As far as the first category is concerned, statechart semantics imply the use of an event pool, which means that there is no loss of event. The concept of the scheduling function and the concept of resource are not clearly defined by the standard, and this is one of the biggest problems for the development of a run-time environment. It is clear that the given definitions are greatly influenced by IEC 1131 and are in contrast to the meaning of the corresponding terms in software engineering. A proposal is to avoid both concepts [33] and use 1) the concept of FB container to partially support the assumed by the resource functionality and 2) the real-time operating system (RTOS) scheduler when scheduling of FB instances is required.

Regarding the event-processing policy, both the first-come order or the priority-based order can be considered. However, to obtain a more efficient response to safety critical events, the priority-based order should be adopted. According to the adopted policy in UML statecharts, the state machine processes one event at a time and finishes all the consequences of that event before processing another event; a policy known as run-to-termination. However, the FB-type definition, and mainly, the definition

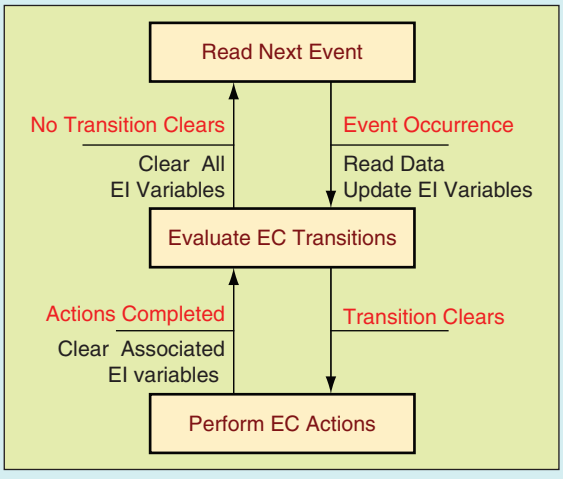

FIGURE 6 – State machine of the active FB instance construct of IEC 61499.

usage of time-sequence diagrams is prescribed for adapters and SIFBs, IEC 61499 does allow their use for the interface definition of any FB type (according to Annex A of IEC 61499-2). Figure 5 shows an example of such an interface behavior description.

This method provides additional means to document the FB behavior to the benefit of both developers and users. Thus, the time-sequence diagrams can be used in the specification phase to define interfaces of new FBs. These specification can be then passed to the FB developer, who implements the functionality of an FB.

Compliance Profiles

One of the main goals of IEC 61499 development was to promote the development of heterogeneous systems composed of control devices of different vendors. Compliance with the standard brings some level of compatibility even if such devices have completely different internals. Quite naturally, the standard cannot foresee upfront all the features of devices' programming, configuration, or communication that need to be standardized. Instead, it defines a flexible and extensible mechanism of compliance profiles.

A compliance profile shall describe how the platform and implementation-specific issues are solved. The structure of a compliance profile is described in IEC 61499-4. In general, an IEC 61499 compliance profile has to define the following three points:

- The portability provisions describe how the models of IEC 61499 can be exchanged between tools of different vendors.
- The interoperability provisions describe how devices from different vendors can communicate with each other.
- The configurability provisions describe how devices from different vendors can be configured and

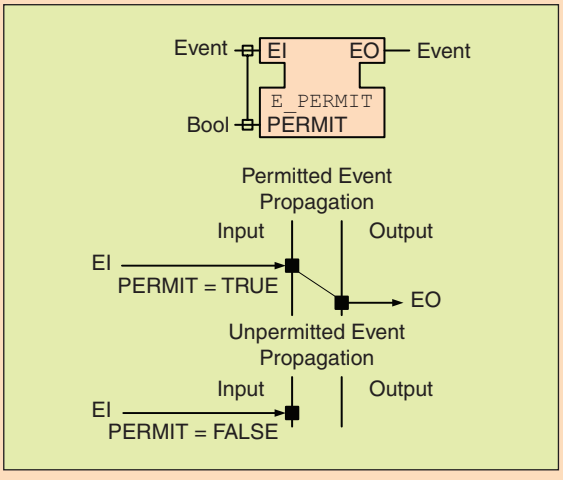

FIGURE 5 – Interface of the standard FB E_PERMIT as defined in Annex A of IEC 61499-1 and the time-sequence diagram describing its behavior.

of EI variables and their use in transition expressions, favors a policy that is influenced by the cycle-based implementation approach, where all pending input events are candidates for processing. Events from the event queue will be transferred to the EI variables that are to be consumed based on their priority.

The problem of the undefined transition evaluation order, which rises when the trigger expressions of two transitions starting from the same state are simultaneously fulfilled, a situation that leads to nondeterminism, can be easily handled by the definition of the evaluation-order priority parameter that explicitly defines the evaluation order on the PSM [33]. The concept of negated trigger event, as defined in [34], can also be used to address this problem. The standard assumes as order of execution the order of transitions in the extensible markup language (XML) specification, but this is a source of ambiguity.

Terms formally defined by the UML 2.0 statechart formalism [30], which is an object-based variant of Harel statecharts, can be used to resolve all the ambiguities in an FB instance execution and obtain a formal model of execution for the FB instance. Such concepts include deferred events, conflicting transitions, firing priorities, completion transitions and completion events, and deferred triggers, which address many of the open issues in FB instance execution. For example, the concept of completion event can be used to formally handle the transition that has as transition condition the expression "1," which is introduced by the standard. Moreover, the concept of deferred event can be used to mark an event in a given state that should not be cleared if it does not trigger any transition in this state.

Many assumptions made by the IEC 61499 community over the last few years regarding the process of defining execution semantics are unsubstantiated without proof of concept, i.e., a reference implementation or a clear theoretical basis. Thus, confusion is created in the domain. For example, authors in [21] claim that a single run of a basic FB is instantaneous or relatively short, and they use it as an assumption to the definition of what they call sequential model of execution. They define as single run the time from the activation of an FB instance until the time that there are no more transitions in the ECC to clear. This assumption is completely arbitrary and, of course, in conflict with the definition of FB as a component that is given by the same authors. There is an assumption in statecharts on zero execution time, but this refers only to the transition time. It should be noted that an issue that differentiates the ECC from the classical statecharts is the fact that the transition label in statecharts

how applications can be downloaded into the devices.

An example of a compliance profile is provided in [21].

The compliance profile concept has the great advantage that the standard can be extended to different needs and also define things the standard has intentionally left open (e.g., the concrete communication between devices on a certain field bus system). However, this flexibility may also bring problems, as it opens the door to vendor-specific extensions that destroy the open distributed system idea of IEC 61499. An example for this is the implementation of ISaGRAF, whose features follow the compliance profile mentioned in [22]. However, the compliance profile is not publicly available, so no other vendor can develop compatible solutions.

In our opinion, a regulatory instrument is needed to keep the control system vendors in line with the ideas of IEC 61499. The international organization O^{3}neida has been volunteering so far to be such a regulatory body. O^{3}neida has been promoting the ideas of IEC 61499 and, on the other hand, has been involved in the development of IEC 61499 compliance profiles [23].

A good example of an issue targeted by a compliance profile is the execution behavior (semantics) of FBs and applications built thereof. The reason for this are weak semantic-related descriptions in IEC 61499-1, which have been interpreted differently by different execution environment developers. This resulted in the situation that the same application can behave differently on different execution environments. Different execution problems were reported in [24]–[26]. To overcome these limitations, O^{3}neida is currently developing a compliance profile defining the execution behavior of run-time environments and also clarifying the ambiguously defined elements of IEC 61499-1 (see [23] for more information).

Many of the semantic ambiguities can be potentially fixed by amending the standard's text. Currently, IEC 61499-1 is in a revision phase. The corresponding IEC working group (of which the authors are members) is actively working on improving several descriptions in the standard to provide a common and clearly defined execution behavior of applications and an FB independent of the underlying execution behavior. Obviously, achieving consensus in the standard would be the best solution to satisfy both the application developers and the device vendors.

Conclusions

Currently, control engineers are challenged by the growing complexity of automation projects,

determines not only the triggering but also event generations. Event generation in ECC is not shown in transition labels, but it is shown in the EC action specification. The term EC action that is used in ECC to define the algorithm and the corresponding event that is to be issued after the execution of the algorithm is probably confusing to the statechart users. It is also clear that the EC action does not have the semantics of the term action as used in statecharts.

The authors in [21] also introduce the parallel hypothesis execution model and claim that it better fits to the not-so-short algorithms assumption, since, as they claim, the execution of each block should take, according to the standard, a short time interval. In [19], the short time execution of algorithms is a prerequisite to assume that a whole daisy chain of event connections between FB instances executes as a critical region. This completely arbitrary and wrong assumption is used to define the nonpreemptive multithreaded resource model of execution, according to which a running FB instance cannot be preempted by another FB instance of the same resource. It should be noted here that it is very important to clearly distinguish the notion of run-to-completion from the concept of thread preemption. Run-to-completion event handling is performed by a thread that can be preempted, and its execution can be suspended in favor of another thread executing the same processing node [30]. The authors proceed in the same article in the axiomatic definition of a number of semantics that are also criticized for not being consistent with the real-time domain concepts of embedded systems and greatly complicate the execution of IEC 61499 design specifications.

Concluding Remarks

The market of industrial systems needs open solutions. Distribution, interoperability, portability, and run-time reconfiguration are open issues that have to be addressed in the development process of today's complex industrial process systems. IEC 61499 is here to provide some solutions. However, it is clear that this standard has been influenced very much from the IEC 1131 FB model and fails in successfully exploiting current software engineering practices. It also has many ambiguities and open issues; the effective use of requirements, the architectural design phase, as well as the execution semantics have to be addressed. This is why a major revision is required for the standard to be seriously considered by the industry. A reliable reference implementation is required to demonstrate the applicability of the standard and also validate the concepts behind it.

Since the standard attempts to introduce at the same time two paradigm shifts, the one from the accompanied by shortened development time and tight quality requirements. New programming methodologies are necessary to increase software quality and reuse. With IEC 61499, modern software engineering methodologies have been adapted to the domain of industrial automation. This investigation showed that IEC 61499 defines several means that can help to improve software quality and reduce the development effort of distributed control systems. However, it also identified some open issues to be solved. How fast these issues can be solved will determine the success of IEC 61499.

Although the current adoption of IEC 61499 is low, it does not determine the failure of IEC 61499. One reason for the low adoption rate so far has been that control engineers needed comprehensive solutions rather than a single technology. Such solutions have finally appeared. The solutions from ICS Triplex, nxtControl, and the open-source project 4DIAC are the promising signs toward broad industrial application of IEC 61499.

Biographies

Alois Zoitl (zoitl@acin.tuwien.ac.at) received his Ph.D. degree in electrical engineering with the focus on automation and control technology from Vienna University of Technology, Austria, in 2007. Since 2002, he has been with the Automation and Control Institute (ACIN), Vienna University of Technology. Currently, he is the head of the Agile Control Group at ACIN. He has participated in several applied research projects with direct relation to industry. His research interests are low-level control of manufacturing systems with the focus on distributed reconfigurable real-time control systems based on IEC 61499. He is a member of the IEC SC65B/WG15 team for the IEC 61499 standard.

Valeriy Vyatkin (v.vyatkin@auckland.ac.nz) is with the University of Auckland, New Zealand. He is the head of infoMechatronics and Industrial Automation research laboratory, whose research focus is the next generation of intelligent automation systems based on self-configuration, plug and play, adaptability, and resilience to faults. His research expertise is software engineering for industrial automation systems, including IEC 61499 architecture, new methods of automatic validation of industrial automation systems, and algorithms improving their performance.

References

[1] Iacocca Institute, "21st century manufacturing enterprise strategy: An industry-led view," Iacocca Institute, Bethlehem, PA, Tech. Rep. D148708, 1991.

[2] *Function Blocks—Part 1: Architecture*, IEC 61499-1, 2005.

procedural to the object based, and the other from the device-centric to the application-centric, a very long time is required for its adoption from industrial engineers. Experiences from the procedural to the OO paradigm shift in the software domain can be exploited to facilitate this paradigm shift.

Acknowledgments

The author is grateful to the anonymous referees for the numerous remarks and suggestions that have led to significant improvements of this article.

Biography

Kleanthis Thramboulidis (thrambo@ece.upatras.gr) received his B.Sc. and Ph.D. degrees in electrical engineering from the University of Patras, Greece, in 1981 and 1989, respectively. He is an associate professor with the Department of Electrical and Computer Engineering, University of Patras, where he is leading the Software Engineering Group. He is currently a visiting professor at Helsinki University of Technology, Finland. He is the designer of CORFU, a framework for the unified development of distributed control and automation systems. He proposed model-integrated mechatronics, a new paradigm for the model-driven development of mechatronic manufacturing systems. His research interests include object technology, model-driven development, service-oriented architectures, embedded systems, and mechatronics. He is a member of the Industrial Electronics Society Technical Committee on Factory Automation.

References

[1] *Function Blocks, Part 1-Part 4*, IEC International Standard 61499, Jan. 2005.

[2] V. Vyatkin, Z. Salcic, P. S. Roop, and J. Fitzgerald, "Now that's smart!" *IEEE Ind. Electron. Mag.*, vol. 1, no. 4, pp. 17–29, Winter 2007.

[3] K. Thramboulidis. (2006). IEC 61499 in factory automation. *Advances in Computer, Information, and Systems Sciences, and Engineering. Proc. IETA 2005*, K. Elleithy. et al., Ed. New York: Springer-Verlag, pp. 115–123 [Online]. Available: http://www.springer.com/engineering/signals/book/978-1-4020-5260-6

[4] A. Zoitl, T. Strasser, K. Hall, R. Staron, C. K. Sünder, and B. Favre-Bulle, "The past, present, and future of IEC 61499," in *Proc. 3rd Int. Conf. Industrial Applications of Holonic and Multi-Agent-Systems, HoloMAS 2007*, Deutschland, pp. 1–14.

[5] K. Thramboulidis and A. Zoupas, "Real-time Java in control and automation: A model driven development approach," in *Proc. 10th IEEE Int. Conf. Emerging Technologies and Factory Automation, ETFA'05*, Catania Italy, Sept. 2005, pp. 38–46.

[6] G. Doukas and K. Thramboulidis. A real-time Linux based framework for model-driven engineering in control and automation. *IEEE Trans. Ind. Electron.* [Online]. Available: http://ieeexplore.ieee.org/xpl/freepre_abs_all.jsp?isnumber=4387790&arnumber=5210159pp. 1–14.

[7] A. Luder, J. Peschke, and M. Heinze, "Control programming using Java," *IEEE Ind. Electron. Mag.*, vol. 2, no. 2, pp. 19–27, June 2008.

[8] S. Panjaitan and G. Frey, "Functional control objects in distributed automation systems," in *Proc. Workshops Intelligent Manufacturing Systems (IMS'07)*, Alicante, Spain, May 23–25, 2007, pp. 293–298.

[3] *Function Blocks—Part 2: Software Tool Requirements*, IEC 61499-2, 2004.

[4] *Function Blocks—Part 4: Rules for Compliance Profiles*, IEC 61499-4, 2005.

[5] K. Thramboulidis, "IEC 61499 in factory automation," in *Proc. Int. Conf. Industrial Electronics, Technology and Automation (CISSE'05—IETA)*, Dec. 2005.

[6] A. Zoitl, T. Strasser, K. Hall, R. Staron, C. Sünder, and B. Favre-Bulle, "The past, present, and future of IEC 61499," in *Proc. 3rd Int. Conf. Industrial Applications of Holonic and Multi-Agent-Systems, HoloMas*, Regensburg, Germany, 2007, pp. 1–14.

[7] A. Zoitl, T. Strasser, C. Sünder, and T. Baier, "Is IEC 61499 in Harmony with IEC 61131-3?" *Ind. Electron. Mag.*, vol. 3, no. 4, pp. 49–55, 2009.

[8] V. Vyatkin, "The IEC 61499 standard and its semantics," *Ind. Electron. Mag.*, vol. 3, no. 4, pp. 40–48, 2009.

[9] R. Lewis, *Modeling Control Systems Using IEC 61499—Applying Function Blocks to Distributed Systems*. London: The Institution of Electrical Engineers, 2001.

[10] V. Vyatkin, *IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design*. Durham, NC: ISA and O^{3}neida, 2007.

[11] *Programmable Controllers—Part 3: Programming Languages*, IEC 61131-3, 1993.

[12] A. Zoitl, *Real-Time Execution for IEC 61499*. ISA and O^{3}neida, Durham, NC: 2009.

[13] ICS Triplex ISaGRAF Inc. (2009, June). *ISaGRAF User's Guide* [Online]. Available: http://www.isagraf.com

[14] D. Lavallée, J.-F. Laliberté, N. Landreaud, K. Thramboulidis, P. Bettez-Poirier, F. Desy, F. Darveau, N. Gendron, and C.-D. Trang. (2009, June). An IEC 61499 configuration with 70 controllers: Challenges, benefits and a discussion on technical decisions [Online]. Available: http://www.isagraf.com/pages/documentation/ETFA07_SS1_Final17Oct2007.pdf

[15] nxtControl GmbH. (2009, June). nxtControl—Next generation software for next generation customers [Online]. Available: http://www.nxtcontrol.com/

[16] Object Management Group. (2009, June). Model driven architecture [Online]. Available: http://www.omg.org/mda/faq_mda.htm

[17] B. Huber, R. Obermaisser, and P. Peti, "MDA-based development in the DECOS integrated architecture—Modeling the hardware platform," in *Proc. 9th IEEE Int. Symp. Object and Component-Oriented Real-Time Distributed Computing, ISORC 2006*, Apr. 2006, pp. 43–52.

[18] 4DIAC Consortium. (2009, June). Framework for distributed automation and control (4DIAC) [Online]. Available: http://www.fordiac.org

[19] C. Sünder, A. Zoitl, J. Christensen, M. Colla, and T. Strasser, "Execution models for the IEC 61499 elements composite function block and subapplication," in *Proc. 2007 5th IEEE Int. Conf. Industrial Informatics*, June 2007, vol. 2, pp. 1169–1175.

[20] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, 2nd ed. New York: ACM Press, 2002.

[21] J. H. Christensen. (2009, June). IEC 61499 compliance profile for feasibility demonstrations [Online]. Available: http://www.holobloc.com/doc/ita/index.htm

[22] TÜVRheinland. (2009, June). ISaGRAF 5.1 assessment according to IEC 61499 [Online]. Available: http://www.isagraf.com/get/ISaGRAF_5-1_1499-TUVconfirmation.pdf

[23] O^{3}neida. (2009, June). Compliance profile [Online]. Available: http://www.oooneida.org/standards_development_Compliance_Profile.html

[24] L. Ferrarini and C. Veber, "Implemenation approaches for the execution model of IEC 61499 applications," in *Proc. 2nd IEEE Int. Conf. Industrial Informatics*, Berlin, Germany, June 2004, pp. 612–617.

[25] K. Thramboulidis and G. Doukas, "IEC 61499 execution model semantics," in *Innovative Algorithms and Techniques in Automation, Industrial Electronics and Telecommunications*, T. Sobh, K. Elleithy, A. Mahmood, and M. Karim, Eds. Berlin: Springer-Verlag, 2007, pp. 223–228.

[26] A. Zoitl and G. Frey, "Special session: S07 execution semantics of IEC 61499 function block applications," in *Proc. 5th IEEE Int. Conf. Industrial Informatics*, June 2007, vol. 2. pp. 1141–1194.

[9] K. Soundararajan and R. W. Brennan, "Design patterns for real-time distributed control system benchmarking," *Robot. Comput.-Integr. Manuf.*, vol. 24, no. 5, pp. 606–615, Oct. 2008.

[10] *Programmable Controllers, Part 3: Programming Languages*, IEC International Standard 61131–3, 2003.

[11] T. M. Egyedi, "Standard-compliant, but incompatible?!" *Comput. Stand. Interfaces*, vol. 29, no. 6, pp. 605–613, 2007.

[12] ISO/IEC Directives—Part 1: Preparation stages for standards [Online]. Available: http://www.iec.ch/ourwork/stages-e.htm

[13] J. Chouinard, D. Lavallée, J. Laliberté, N. Landreaud, K. Thramboulidis, P. Bettez-Poirier, F. Desy, F. Darveau, N. Gendron, and C. Trang. (2007). An IEC 61499 configuration with 70 controllers; challenges, benefits and a discussion on technical decisions [Online]. Available: http://www.isagraf.com/pages/documentation/whitepapers.htm

[14] J. Bergin and R. Winder. (2000). Understanding object oriented programming [Online]. Available: http://csis.pace.edu/~bergin/patterns/ppoop.html

[15] K. Thramboulidis, "A constructivism-based approach to teach object-oriented programming," *J. Informat. Educ. Res.*, vol. 5, no. 1, pp. 1–12, 2003.

[16] K. Thramboulidis, "Using UML in control and automation: A model driven approach," in *Proc. 2nd IEEE Int. Conf. Industrial Informatics, INDIN'04*, Berlin Germany, June 24–26, 2004, pp. 587–593.

[17] K. Thramboulidis, S. Sierla, N. Papakonstantinou, and K. Koskinen, "An IEC 61499 based approach for distributed batch process control," in *Proc. 5th IEEE Int. Conf. Industrial Informatics (INDIN 07)*, Vienna, Austria, July 23–27, 2007, pp. 177–182.

[18] *Representation of Process Control Engineering Requests in P&I Diagrams and Data Exchange Between P&ID Tools and PCE-CAE Tools*, International Standard IEC 62424, 2005.

[19] C. Sunder, A. Zoitl, J. H. Christensen, V. Vyatkin, R. W. Brennan, A. Valentini, L. Ferrarini, T. Strasser, J.L. Martinez-Lastra, and F. Auinger, "Usability and interoperability of IEC 61499 based distributed automation systems," in *Proc. 4th IEEE Int. Conf. Industrial Informatics (INDIN 06)*, 2006, pp. 31–37.

[20] V. Vyatkin and V. Dubinin, "Sequential axiomatic model for execution of basic function blocks in IEC 61499," in *Proc. 5th IEEE Int. Conf. Industrial Informatics (INDIN 07)*, Vienna, Austria, July 23–27, 2007, pp. 1183–1188.

[21] V. Vyatkin, V. Dubinin, C. Veber, and L. Ferrarini, "Alternatives for execution semantics of IEC 61499," in *Proc. 5th IEEE Int. Conf. Industrial Informatics (INDIN 07)*, Vienna, Austria, July 23–27, 2007.

[22] V. Vyatkin and V. Dubinin. (2007, Oct. 3). Execution model of IEC 61499 function block based on sequential hypothesis [Online]. Available: http://www.ece.auckland.ac.nz/~vyatkin/o3fb/vd_seqsem.pdf

[23] K. Thramboulidis, "A model based approach to address inefficiencies of the IEC 61499 function block model," in *Proc. 19th Int. Conf. Software and Systems Engineering*, Paris, France, Dec. 2006.

[24] J. P. Peltola, J. H. Christensen, S. A. Sierla, and K. O. Koskinen, "A migration path to IEC 61499 for the batch process industry," in *Proc. 5th IEEE Int. Conf. Industrial Informatics INDIN 07*, Vienna, Austria, July 2007, pp. 811–816.

[25] C. Szyperski, "Component technology—What, where, and how?" in *Proc. 25th Int. Conf. Software Engineering (ICSE'03)*, p. 684.

[26] C. Szyperski. (1999). Components vs. objects vs. component objects [Online]. Available: http://www.oberon2005.ru/paper/cs1999.pdf

[27] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*. Englewood Cliffs, NJ: Prentice-Hall, 1996.

[28] J. Ivers, P. Clements, D. Garlan, R. Nord, B. Schmerl, and J. Silva, "Documenting component and connector views with UML 2.0," Tech. Rep. CMU/SEI-2004-TR-008 ESC-TR-2004-008, Software Eng. Instt., Pittsburgh, Apr. 2004.

[29] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*, 2nd ed., Reading, MA: Addison-Wesley, 2005.

[30] OMG, "Unified Modeling Language: Superstructure, version 2.1.1," formal/2007-02-03, Needham, MA.

[31] R. Soley and the OMG Staff Strategy Group. (2000, Nov. 27). Model-driven architecture. White Paper, Draft 3.2 [Online]. Available: ftp://ftp.omg.org/pub/docs/omg/00-11-05.pdf

[32] B. Selic and L. Motus, "Using models in real-time software design," *IEEE Control Syst. Mag.*, vol. 23, no. 3, pp. 31–42, June 2003.

[33] K. Thramboulidis and G. Doukas, "IEC 61499 execution model semantics," in *Proc. Int. Conf. Industrial Electronics, Technology and Automation (CISSE-IETA 06)*, Dec. 4–14, 2006, pp. 223–228.

[34] M. von der Beek, "A comparison of statechart variants," *Formal Techniques in Real-Time and Fault-Tolerant Systems* (Lecture Notes in Computer Science, vol. 863), L. de Roever and J. Vytopil, Eds. Berlin: Springer-Verlag, pp. 128–148.