The Function Block Model in Embedded Control and Automation From IEC61131 to IEC61499

KLEANTHIS THRAMBOULIDIS Electrical & Computer Engineering University of Patras 26500 Patras GREECE thrambo@ece.upatras.gr/thrambo

Abstract: - The Function Block (FB) model was first standardized by the 1131 standard of the International Electrotechnical Commission (IEC) for programmable controllers. This standard was successfully adopted by the industry but it seems to have several constraints for the development of today's complex embedded control and automation systems. These constraints are mainly imposed by the procedural programming paradigm and the device centric approach that are adopted by the standard. The IEC to address these constraints proposed the 61499 standard that is an attempt to exploit object-orientation and the application-centric paradigm in the control and automation domain. In this paper, the FB models of 1131 and 61499 are briefly described and several unclear issues related to the programming paradigms adopted, interoperability, composability and execution semantics of these FB models are clarified. The paper focuses on the execution semantics of the formation standard since this is one of the most important reasons that the industry has not yet accepted this standard.

Key-Words: - embedded control and automation systems, IEC 61131; IEC 61499; 1131 Function Block Model; IEC61499 execution environment; execution model semantics; Factory Automation.

1 Introduction

The IEC61131 standard [1] was an attempt to unify, at least at the semantic level, the main types of languages used in practice for PLC programming around the world [2]. The standard that was published in 1993 defines among the five languages the Function Block Diagram which established the so called Function Block (FB) model in the industrial control programming domain. The 1131 FB is based on the procedural programming paradigm and promotes the device centric approach the development of industrial in systems development. A great push to the adoption of the FB model from industry was given by the PLCOpen association that was created to promote the usage and supply of products in conformance with the 1131 standard [3].

However, the increased complexity of embedded systems in the control and automation domain cannot be effectively addressed by the procedural and device centric paradigms. As Lewis states in [4] "There are a number of limitations with the original function block concept introduced by the (...) 1131. With the (...) (FBD) graphical language, function blocks can be linked by simply connecting data flow connections between block inputs and output variables."

Software engineering has to demonstrate significant progress with technologies such as object and component technology and model driven engineering that can be exploited to improve the development process of embedded control and automation systems. To address today's challenges in industrial automation systems development, the International Electrotechnical Commission (IEC) has defined the 61499 FB model [5] as an extension to the 1131 FB. This was also an attempt of the IEC to "open" the industrial systems market and meet among others, requirements such as interoperability, portability. distribution, agility. run-time reconfigurability, higher availability and reliability.

The new model is assumed to introduce a paradigm shift from the procedural approach adopted by the 1131 FB model to the object oriented one and also a shift from the device to the application centric approach. However, it is clear that this standard has been influenced very much from the 1131 FB model and fails in successfully exploiting current software engineering practices. Even though it has been officially accepted by the year 2005 it is not yet

adopted by the industry and its status in the academic research community is questionable.

In this paper, an attempt to clarify a number of unclear issues on these FB models is done. More emphasis is given to the execution semantics of these models since the open issues in the execution semantics of the 61499 is probably the most important reason for the fact that the industry has not yet adopted this standard. The ongoing discussion for enhancing the 1131 FB model to support the OO paradigm is discussed and compared to the 61499 approach.

The remainder of this paper is organized as follows. In the next 2 sections a brief introduction to the 1131 and IEC61499 function block models is given. In section 4, the main important aspects of the 1131 and 61499 are discussed. In section 5, the execution semantics of the FB model are discussed, with more emphasis on IEC61499. Finally, the paper is concluded in the last section.

2 The IEC61131 Function Block Model

Figure 1 presents in terms of a semi formal UML model the basic constructs of the 1131 FB model. It also captures the main concepts defined by the 1131

standard that are required for the specification and execution of 1131 based applications.

The term configuration is used to refer to the organization of the software that solves a specific control problem. A configuration is usually used to specify the control application that is executed in one PLC. However, for complex control problems the control application will be defined as an aggregation of configurations running on separate PLCs. A configuration is executed on a network of interconnected devices that are usually PLCs. Each device has one or more processing units, and each unit normally has one resource but it may also have more resources, as shown in fig. 1. The resource provides the services, i.e. the infrastructure required for the execution of 1131 programs. One of the most important services provided by the resource is the functionality required to implement the interface of the software with the physical I/O channels of the PLC.

3 A Brief Introduction to the IEC61499 Function Block Model

The 61499 FB was defined as an extension of the 1131 FB to address today's challenges in industrial automation systems development. The FB is defined



Fig. 1. The IEC 1131 Function Block Model.

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 fig. 2. 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.



Fig. 2. Graphical representation of the FB type design level construct.

The FB is more than an object since it proceeds one step further and defines a specific way of capturing the dynamic behaviour of the object that represents. It proposes the use of a specific kind of statechart that is called Execution Control Chart (ECC) to specify the dynamics of the object. An ECC consists of EC states, EC transitions and EC actions, as shown in fig. 3. An EC state may have zero or more associated EC actions, except from the initial state that shall 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 one state to another. An EC transition has an associated Boolean expression that may contain event inputs, data inputs, and internal variables. As soon as this expression becomes true the EC transition fires.

An application is defined as a network of interconnected FB instances that accepts inputs from the mechanical system, through sensors, and generates outputs that are sent to the mechanical system through actuators. Fig. 4 presents the basic constructs of the 61499 FB model in terms of a semi formal UML model.



Fig. 3. Execution Control Chart (ECC) of the PID_SIMPLE Function Block type.

The great influence of the 1131 on the definition of the IEC61499 standard can be easily identified. Even though an attempt was done to exploit current software engineering practices, the specification has many disadvantages regarding its theoretical basis in exploiting current software engineering concepts and technologies, such as object orientation component and based development. This is probably one of the most important reasons for the many ambiguities that exist in the specification. Unfortunately there was no concept evaluation process in the form of a reference implementation before the acceptance of the standard [6]. FBRT (www. holobloc.com), 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 the semantics defined by the specification. Other implementations were also not used to revisit and resolve ambiguities in the standard.

Unfortunately even though a lot of papers have been published on the subject, the number of actual implementations, even prototypes, is very limited. There is no mature reference implementation to demonstrate the applicability and the advantages of the specification. Several prototype or under development IDE's exist to support the development process. FBDK (www.holobloc.com), CORFU/ Archimedes (http://seg.ece.upatras.gr/seg/dev/iec 61499.htm) [30], and 4DIAC (www.fordiac.org/) are currently the most known in the community. There are also several prototype run-time environments such as FBRT, Archimedes RTSJ-AXE (http://seg. ece.upatras.gr/mim/RTSJ-AXEpackage.htm), Archimedes RTAI-Linux, Forte (http://source forge.net/ projects/fordiac), etc. A small number of example applications have been developed to demonstrate the applicability of the specification (see for example http://seg.ece.upatras .gr/seg/dev/iec61499.htm)



Fig. 4. The IEC 61499 Function Block model.

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 promising view is the Academic view while the disappointing view is the Industry's view [6]. It can be stated that there is a tendency from industry to reject the IEC 61499 standard simply because: a) its learning curve is perceived as being very steep, and b) there is no mature reference implementation to demonstrate the applicability and also the advantages of the new specification. There are also a number of nontechnical reasons for industrial engineers for not adopting the standard [31].

4 A Discussion on 1131 and 61499 Features

The following statement/question was raised during the industry day of ETFA 08 conference that was devoted to the 1131 and 61499 standards [33]. The question that was the motivation for this paper is the following: "I am working for many years with 61499. I know very well the concept behind it but at this moment I will try to make a question from the control Engineer's point of view. Attending the morning session on 61499 I was able to hear from the presenters that this standard addresses all the problems imposed by 1131 and mainly those of

portability, *interoperability*, distribution and reconfigurability. Attending the evening session I was also able to see the presenters to demonstrate the support of 1131 to portability, interoperability, distribution and reconfigurability. I was also able to see the slide of the car composition that was used from PLCOpen to focus on the support of 1131 to the component based approach. Even more I attended with great interest the proposal (from Codesys) for extending the 1131 to support the OO paradigm. What I can say from my experience working with 61499 all these years is that 1131 with such an extension will be several steps in front of 61499. In fact, I am a little confused about the need for both standards in the case of the 1131 extension to OOP. The control Engineer is much more confused trying to understand what is the way to follow for the next generation of automation embedded systems".

In this section the main properties of these standards are discussed in detail in order to provide answers to the questions raised by the above position statement. The presented in [32] object-oriented extension to the 1131 FB model is not taken in account in this discussion, since this extension has not yet been accepted by the 1131 working group. However, it should be noted that from the presentation of CoDeSys in [33] it has been clear that with the new extension the 1131 will provide a much better support for the OO paradigm than the one provided by the 61499.

4.1 Interoperability and Portability

Both models claim to support interoperability and portability. However, a distinction has to be done between edit-time and run-time interoperability and XML-based portability. PLCOpen its with specification for 1131 provides a good support for edit-time interoperability and portability but these properties are not supported for the run-time. Eventhough 61499 has addressed both issues as far as the edit-time is concerned, it fails in both for the run-time. By using the term "interoperability" both communities, i.e. the 1131 and the 61499, refer to the edit- time interoperability, that is the ability to exchange design-time model elements such as FB types and FB networks between development tools. This is also the case for the use of the term "portability". The run-time interoperability, i.e. the possibility of several applications running on different hardware and software infrastructures to interoperate is not addressed by both standards. This is also the case for the run-time portability.

4.2 Monolithic vs. Component-based Approach

The term "run-time component" is used in software development from 1988 [9]. A component based application can be considered as a network of run-time components. The term "network of run-time components" can be used to refer to the control application to emphasize that it is not monolithic. "Network" is, according to Collins Cobuild dictionary, a system of things which are connected and which operate together. According to this definition a network of run-time components is a system of run-time components which are connected and which operate together.

Due to the misuse of the term component, the difference between the monolithic and the component based approach in both communities is not clear. The 1131 community adopts the monolithic approach regarding the binary of the application. This is also the case for the majority of the 61499 tools; only a restricted number of tools support the component-based approach for the execution of IEC61499-based control applications [11]. Reconfigurability of the automation system is greatly depended on this characteristic of the application. In fact run-time reconfigurability can be applied only in the component based approach.

The component-based approach has several advantages compared to the monolithic one. As stated in [7] "Unfortunately, MathWorks products tend to generate monolithic code rather than component-based code. This makes it more difficult to validate or update the code." As claimed in [10], component-based development is "a solution that has a long tradition of advocates, is recommended by leading experts, and is quickly gaining support." In the same paper it is also stated that: "software components support modular engineering practices, just as integrated circuits support modular design of hardware", "Component-based development has appeal in distributed exceptional software development" and "the nature of components forces designers and developers to better encapsulate functionality into cohesive, reasonably welldocumented chunks of software." The advantages of the component-based design of control systems are also discussed in [7].

An application can be considered monolithic or component based either in the source or the binary level. According to Szyperski [8], a software component has to be a unit of deployment and thus it has to be an executable deliverable for a (virtual) machine, so no human intervention will be required for its use. Adopting the above definition of component, a monolithic application, in the binary level, is an application that its run-time is a single piece of executable code. The component-based application is the application whose run-time is considered as an aggregation of interconnected binary (run-time) components.

According to the distribution model described in [5] "an *application* or *subapplication* can be distributed by allocating its *function block instances* to different *resources* in one or more *devices*." The standard [5] also states that "a function block must form an atomic unit of distribution."

From the above and the definition of component given by Szyperski [8], it is evident that the objective of the IEC61499 is to move from the monolithic application approach to the promising component-based one, even though several researchers claim that this is not true. The remainder of this subsection argues on this direction. The runtime support for composability is one of the significant contributions of the IEC61499 compared to the IEC61131 function block model. Design-time composability is a feature already supported to a great extend by IEC61131 and widely used by industrial engineers for many years.

Taking into account that a monolithic application, in the binary level, is an application that its run-time is a single piece of executable code and the following statements of the IEC61499 standard [5]:

- 1. the definition of the function block as an atomic unit of distribution ("a function block must form an atomic unit of distribution"), and
- 2. the description of the management function blocks that provide functionality for application management to "create, initialize, start, stop, delete, query the existence (...) of data types, function block types and instances and connections among function block instances",

it is more than evident that the IEC61499 favors the shift from the monolithic application approach to the promising component-based one. Of course, in this case the function block is also used as design time artifact and thus the developer exploits the advantages imposed by this. The functionality of the management function blocks to "create, initialize, start, stop, delete, query the existence (...) of data types, function block types and instances and connections among function block instances" [5] can be exploited only in the case of the component-based approach when the application's run-time is considered as an aggregation of interconnected binary (run-time) function block instances. This functionality has no meaning in the case of a monolithic (in the binary level) application.

The following is the exact statement from the standard [5] (p. 45) regarding the functionality provided by the management function blocks. "Extending the functional requirements for

"application management" in (...) to the distributed application model of this part of IEC 61499 indicates that *services* for management of resources and applications in IPMCSs should be able to perform the following *functions:* 1. In a *resource*, create, initialize, start, stop, delete, query the existence and *attributes* of, and provide notification of changes in availability and status of: data types, function block types and instances, and connections among function block instances."

4.3 The Device Centric vs. Application Centric Approach

The 1131 has no support for distribution and is mainly used with the device centric approach. At the time the developer designs the application (2nd phase in fig. 5) the system layer (network of devices) has already been developed so he knows in detail the target of each sub-system and also the channels information.

According to the application centric approach, the application is designed before the definition of the system layer as a network of interconnected devices, as shown in fig. 6. Device related info, as for example I/O channels information, is not available to the developer during the application design time. Platform Independent modeling that is one of the core issues of Model Driven Engineering assumes the application centric approach.



Fig. 5. The device centric approach.



Fig. 6. The application centric approach.

4.4 Distribution Support – the Service Interface Function Block

To address the distribution problem in the devices that constitute the run-time environment for the application, the 61499 standard introduces the concept of the Service Interface Function Block (SIFB) as a "function block which provides one or more services to an application, based on a mapping of service primitives to the function block's event inputs, event outputs, data inputs and data outputs." In this way, as it is claimed in [20], the standard defines how data and event connections of the FB diagram should be implemented. The use of the SIFB in the design diagram complicates the FB network diagram, completely destroys location transparency and makes it dependent on a specific configuration of the target platform. All the design alternatives except the one described in [20] adopt the use of SIFBs in the design level.

A better approach that provides distribution flexibility and favors location transparency is described in [20]. According to this, two layers the mechanical process interface (MPI) layer and the IPCP layer have been defined to provide a set of services that have to be provided by the execution environment and used by the ESS and the devices to automatically setup and implement both the event and data connections with the mechanical process and the other devices where applications components have assigned. Actually, the MPI provides the communication infrastructure and the abstraction required by the control application to interface with the controlled mechanical process. This approach:

- a) simplifies the FB design diagrams,
- b) de-couples the FB design diagrams from the physical architecture, and
- c) results in a more flexible reconfiguration process that is required during the operational phase.

The SIFBs are used by the standard to implement the mapping of the application event and data (e.g. Tank1.highTempAlarm, Tank1.temp) to the Mechanical Process (MP) parameters (e.g. high temperature alarm of "Tank1" tank of the MP, temperature of "Tank1" tank of the MP) that are I/Os of the control application. This is done by implementing the SIFBs on top of the "process interface" as defined in the standard [5, fig. 3]. This means that the SIFB developer has to implement the mapping of the events and data of the SIFB to the corresponding MP parameters using the Application Programming Interface (API) of the processinterface layer.

The MPI of the run-time environment is a layer on top of the process-interface layer (as is used by the standard). MPI provides a parameterized functionality to map the events and data of the application to the corresponding MP parameters using the services of the process-interface layer. The MPI is an alternative more flexible way to provide the same functionality that has to be implemented by the SIFB developer. The definition of MPI is an attempt to increase the level of abstraction on which the application designer is working. With this abstraction the application designer has to refer to the MP parameters by their specific abstract identifiers, which may be names, as for example Tank1.highTempAlarm, instead of either implementing the required SIFB for the specific process-interface or looking for a commercial-of-theshelf (COTS) SIFB that satisfies the application needs and conforms to the target platform. The configuration of the MPI to provide this layer of abstraction to the application designer is a job that should be performed at the device configuration phase.

Since the MPI is a layer on top of the processinterface layer, the proposed architecture of the runtime environment provides to the application developer the following alternatives:

- Use the MPI layer to interface with the MP and avoid the use of the SIFBs;
- Use the SIFBs on top of process-interface layer as is defined by the standard.

Of course an alternative may be to use the concept of the SIFB on top of the MPI, but this is not an effective design decision.

As far as the argument used by several researchers that SIFBs are needed "to have a consistent description of a system in uniform terms of function blocks", we claim that it is more productive for the developer to work on an upper layer of abstraction that hides communication infrastructure details (as is also the case for processing infrastructure details). Working on an upper layer of abstraction the designer will concentrate only on the definition of the application logic. This approach is adopted in every component and model driven engineering approach [21] and of course it is one of the objectives of the presented framework. Authors in [21] claim that in order "to facilitate traceability, reuse, and evolution, systems should be specified as compositions of clearly separated and separately specified concerns of interest". The definition of the MPI layer is analogous to the middleware layer and allows the developer to apply vertical separation of concerns through the use of the platform independent model (PIM) and the platform specific model (PSM) in the development process. Following this paradigm the developer applies separation of concerns by defining first a MPI-technology independent model of the application and then a MPI-technology dependent model. The vertical separation of concerns in the

form of PIM and PSM reduces complexity through abstraction and the horizontal separation of concern reduces complexity by describing the system using manageable system views [22].

The term "separation of concerns" was introduced by E. W. Dijkstra in [23]. As Reade claims in [24], the programmer has to do "several things at the same time, namely, 1. describe what is to be computed; 2. organise the computation sequencing into small steps; 3. organise memory management during the computation." In today's distributed systems one more issue can be added into the third bullet, i.e., "organize communication management during the computation".

Reade claims in [24] that the programmer should be able "to concentrate on the first of the three tasks (describing what is to be computed) without being distracted by the other two, more administrative, tasks. Clearly, administration is important but by separating it from the main task we are likely to get more reliable results and we can ease the programming problem by automating much of the administration. The separation of concerns has other advantages as well. (...) Furthermore, descriptions of what is to be computed should be free of such detailed step-by-step descriptions of how to do it if they are to be evaluated with different machine architectures."

A layered design as the one adopted in [16] is one way to apply the concept of "separation of concerns" and have the industrial designer exploit its advantages.

There is an argument used by the 61499 community on the use of the SIFBs that is stated as follows: "Communication SIFBs are added at a later stage of the application development namely at the mapping stage. This is the stage when application parts are mapped to the control devices. Therefore a not mapped IEC 61499 application does not consider the hardware which is very important for developing distributed control systems."

It is clear that the mapping process as described and implemented by the 61499 is not user friendly, not effective since it implies the redesign of the whole Function Block Network (FBN) by introducing several SIFBs with many new event and data connections. This process further complicates the FBN and makes it platform specific. And of course the developer has to work for any change on this complicated and platform specific FBN, either this change concerns the introduction of a new FB instance or the re-assignment of an FB instance to a new device (due to a change in the network of devices). A much simpler process of mapping can be obtained adopting a run-time environment that automatically creates the interconnections between FB instances that are located on different devices using the appropriate services of the run-time environment.

4.5 The Procedural vs. the OO Approach

There is a trend in the 1131 community to consider the 1131 FB as an object or component using the argument that it has "a strong encapsulation concept". Actually it has a strong encapsulation concept but this is not enough to classify the 1131 FB as object or component. The function and the procedure also encapsulate their implementations but they are not objects. A detailed comparison of 61499 with the procedural approach is given in [18].

As Lewis states in [4] "With the ... (FBD) graphical language, function blocks can be linked by simply connecting data flow connections between block inputs and output variables (...). Each function block (1131) provides a single internal algorithm that is executed when the function block is invoked." Objects accept messages and provide several operations to handle the various messages that may accept. According to Booch [19] objects "exist in time, are changeable, have state, are instantiated, and can be created, destroyed and shared"

Visual assembly tools, as those that support 1131, are used to assemble objects, but each one of these objects represent just a process that has to be executed on the input data. Moreover, an object based system should support the implementation of a system design that is based on the concepts of class and object. IEC 1131 cannot be used to implement an object oriented design. The 1131 function block cannot be used to realize an object type (class) with name Valve, having attributes and operations, such as open() and close(), not an object type with name ElevatorCabin, having operations such as moveUp(), moveDown(), stop(), etc.

Since the 1131 is considered as procedural approach while the 61499 object-based, the most important issue that has to be addressed is to find ways to make the paradigm shift that is required easier for the industrial engineer. Industrial engineers are familiar with the device-centric and procedural based paradigm that is adopted by current practices in industrial systems development. These paradigms are also adopted by the widely used by industry 1131 standard.

It is clear that the 61499 FB model is not only a new technology in the domain but it imposes a paradigm shift. The new technology is based on the application-centric approach and also adopts the object-oriented approach. 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 object-oriented paradigm. This is due to the fact that this shift should also be accompanied by the device-centric paradigm to the application-centric one.

5 Execution Semantics

Both standards suffer from the absence of well defined execution semantics. This is one of the most important reasons that they do not support portability. In 1131 the normal execution order of FBs in a FB network is determined by the function block dependency on the other FBs. The order "normally runs from left to right because blocks to the right depend on the output values of the blocks on the left" [4]. However, when a feedback path is introduced "the execution order cannot be determined from the diagram, since the execution of both blocks depends on an output value of the other block" [4]. In a complex network it is very difficult, if not impossible, for a run-time environment to determine a valid order of execution. "As a consequence, an important aspect of a function block network, i.e. the method for defining the execution order of blocks, is not consistent or portable across control systems" [4].

5.1 IEC 61499 FB Model Execution Semantics

The IEC61499 was assumed to address the above problems of 1131 with the execution semantics of the FB network using the concept of event and the event connection. Two main kinds of FB types are proposed by the standard, the basic FB type and the composite FB type. The basic function block type utilizes the ECC to control the execution of its algorithms. The composite function block type is composed of a network of interconnected FB instances and has no ECC, so its execution semantics are quite different from those of the basic FB type.

According to the standard [5] the execution of algorithms in the basic FB instance is "coordinated by the execution control portion (FB head) of the FB instance in response to events to its event inputs." Fig. 7 presents the time related characteristics of the execution logic of a basic FB instance as defined by the standard. t2 is the time that the event arrives at the event input of the FB instance and the ECC starts its execution. It is assumed that at a previous time t1, the required by the FB instance data in order to process this event were made available. At t3 the execution control function notifies the scheduling function to schedule an algorithm for execution. At t4 the execution begins and at t5 the algorithm derives the output data that are associated with the WITH qualifier to the output event of the corresponding EC action. At t6 the scheduling function is notified that the algorithm execution has ended. The scheduling function invokes at t7 the execution control function, which signals at t8 the event that is defined by the corresponding EC action.

It is evident that the standard assumes the existence of a scheduling function in the associated 61499 resource. However, for devices with resource constraints such as IEC-compliant sensors and actuators a scheduler not only implies a big overhead but it is actually not required. Moreover, for devices with no restrictions on resources, it is claimed in this section that this scheduler is not actually required, since the thread that executes the ECC can also execute the algorithms of the corresponding EC actions. This thread can be either the thread of the FB instance in the case of an active FB instance (FB instance with its own thread of execution) or the thread of the FB container [16] in which the FB instance was injected.



Fig. 7. Execution model of Basic Function Block [5]

In the case of assigning the same thread for the execution of the ECC and algorithms, that is the case of our execution environments [16][17][26], it is clear that the ECC cannot react during the execution of algorithms to the events that occur at the FB instance's event inputs. However, this is not possible even for the case of having two threads, one for the ECC and one for algorithms, that is the one proposed by the standard, since according to [5] "all operations performed from an occurrence of transition t1 to an occurrence of t2 (see fig. 4) shall be implemented as a critical region with a lock on the function block instance."

To further examine this problem, the operation state machine of the ECC presented in fig.8 is used.

S0 represents the idle state, S1 represents the state of evaluating transitions and S2 the state of performing the actions. Based on this state machine the following two scenarios are considered:

- 1. the event has to be consumed by the FB instance before the occurrence of the next event to its event inputs. That is, the transition t2 should occur before the arrival of the next event,
- 2. the event may occur when the FB instance is in one of the states S1 or S2.



Fig. 8. ECC operation state machine [5].

To satisfy the requirement of the first scenario the FBN should be scheduled in such a way that the execution of the FB instance will be terminated before its deadline that should be before the appearance of the next event. For the second scenario, if the loss of the event is permitted by the nature of the application, the event is simply ignored, either wise the event is stored so as to be consumed immediately after the transition t2 to the S0 state. All the above alternatives may be supported by the execution environment given the appropriate notation at the design level. For example, the control engineer should define, at design time, for each event the following properties: 'event loss permitted' and 'event consumption before next event'. The latter property will be utilized during schedulability analysis of the FBN to define the deadline of the corresponding FB instance that has to be met by the scheduler.

The solution proposed above and implemented in the context of RTAI-AXE [11] and RTSJ-AXE [17] execution environments can also implement the proposed by the standard behaviour, if there is a need for such behaviour. After the execution of the ECC the corresponding thread should issue a yield command to the operating system that will result to the rescheduling of this thread, which of course in this time will execute the algorithms of the associated EC actions. If a different priority for the algorithm execution is required the proper update of the thread's priority is required before the yield operation.

A different approach is proposed in [27] where two threads are used for the execution of FB instance: a) the "event executing" thread, which handles incoming events and execute the ECC, and b) the "algorithm executing" thread, which executes the activated algorithms. This approach was adopted, according to the authors, to allow the acceptance of events by the FB instances during algorithm execution. However, this doesn't really make any sense if we consider the constraint imposed by the FB model according to which the new incoming event(s) should not trigger an ECC transition before the currently executing FB algorithm/action finishes. The only advantage of this approach i.e., the ability to execute FB algorithms and ECC with deferent priorities can be also obtained in the case of one thread as it was already stated above.

A detailed discussion on the execution model semantics including those of the composite FB type and the FB network can be found in [28]. Implementation model alternatives related to execution semantics for the FB network are presented and discussed in [29].

5.2 The Sequential Hypothesis Execution Model

From the above it is clear that there are many open issues in the execution semantics of 61499 and this is one of the main reasons for the industry has not yet adopted the standard. Many assumptions made last years in the 61499 community in the process of defining the execution semantics are questionable, without proof of concept, which could be either a reference implementation or clear theoretical basis, and thus create confusion in the domain. The most important is the one promoted by OOONEIDA (www.oooneida.org) and defined by the Workgroup on Execution Models of IEC 61499 Function Block Applications. (http://www.oooneida. org/standards development Compliance Profile.ht ml). This model is defined in a form of axiomatic definition based on a set of 6 postulates. It is called sequential hypothesis execution model and it has a great influence on the 61499 community. It is claimed that the sequential model is expected to be immediately applicable, implemented in a number of software tools. This model is disputed in this section for its correctness and its clear theoretical basis. It is argued that it greatly complicates the execution of 61499 design specifications and it is criticized for not been consistent with the real-time domain concepts of embedded systems.

In [12], the distinction between the instantaneous occurrence of an event and its handling in not clear. Authors use arguments as, "So there is no such thing as "clearing an event" because it is never "set". One might think of the event as a single pulse on the transition line..." to form postulate #4 which states that "Event input of a function block clears after single ECC transition, regardless of was this event used in the evaluation or not". The definition of so

misty postulates can only lead the community to confusion. Semantics of the UML 2.0 state machine [13] such as deferred event, completion transitions and completion events provide very clear answers to the questions postulate #4 is assumed to address. The answer for example to the question "How long an event is alive?" is very clearly given by the following statement: "An event that does not trigger any transitions in the current state, will not be dispatched if its type matches one of the types in the deferred event set of that state. Instead, it remains in the event pool while another non-deferred event is dispatched instead" [13]. There is also an alternative to allow the designer to define an event in the triggering expression as non-consumable even for the case it triggers the transition [14]. This seems to be the most expressive solution if combined with the one adopted in [13].

Postulate #5 that is "Output events are issued immediately after the corresponding action is completed" has nothing to add to the standard according to which "The algorithm (if any) associated with an EC action, and the event (if any) to be issued on completion of the algorithm, ..." The same is true for postulate #6 that is defined as "If a function block emits several output events in one state of ECC, they are emitted sequentially", since according to the standard the event of each EC action is issued on completion of the algorithm and the EC actions of the current state are executed sequentially. The authors claim that postulate #6 "implies the execution model which we further refer to as "Sequential hypothesis"". They also claim that "Both these postulates", i.e. #5 and #6, "imply that there is no such thing as concurrent execution of function blocks within a single resource, or pre-emption of one block by another." This is an arbitrary and wrong statement. Actually it should be possible to execute a FBN either concurrently or not and this has not to be defined by the standard as stated in [15] "we believe that it does not accord well to the letter and the spirit of the standard." The sequential or concurrent execution of FB instances has to be transparent to the control engineer. Concurrency should be used as a means to meet stringent real-time constraints that the designer has to specify on the design model [16].

Postulate #2, which states that "Execution (a single run) cannot be pre-empted by execution of another function block (in the same resource)." is completely arbitrary even though it appears to be a consequence of the following statement of the standard "All operations performed from an occurrence of transition t1 to an occurrence of t2 shall be implemented as a critical region with a lock

1606

on the function block instance." This last statement is not valid for the case of active FB instances; it is legal only for the case of a passive FB instance that can be executed in the context of two at least external threads. However, even in this case, preemption is not allowed only for the threads competing to run the passive FB instance. Any other thread may preempt the currently running thread if it has higher priority or if the time slice of the running thread has expired. The authors also claim that "Since (according to #2) FB0 cannot be pre-empted by FB1, the EO1 needs to be stored." However, the handling of events is done by the OS. When an event is issued, the event is dispatched to the consumer's task queue, placing the consumer task to the queue of the OS that is dedicated for the ready for execution task. It is the task of the OS scheduler to decide if the current task (the one that issued the event) will continue its execution or be preempted by the consumer of this event task or another task. So, if FB1 has been assigned at the design time a higher priority than FB0, as for example is the case when FB1 closes a control loop of very high priority, FB0 will of course be preempted by FB1.

Postulate #1 is also arbitrary and constrains the implementations of efficient run-time environments. It is not valid for the case that the thread that implements the active FB instance is defined as periodic as is the case for the Archimedes RTSJ-AXE [17].

Authors in the context of sequential hypothesis claim that "It is needed to ensure the processing of input events will follow the order of their occurrence" [15]. However, this hypothesis completely destroys the event based paradigm of computation. It provides no way of handling higher priority events generated from highly time constraint procedures in the mechanical process such as emergency alarms. This is clear from the statement of the authors "Queue 2 is not empty and adds the request of kind $r \rightarrow S$ to the Queue 1,". This can only be partially legal if FB types are used to represent just functions, but this is not the case for the IEC61499. And of course, if the order of execution can be completely defined by the design diagram as they explain, the question is: what is the reason for using the entity that they call 'scheduler'? What authors describe as 'Sequential hypothesis' with all this described behavior can be supported with more simple and formal constructs provided by the OSs.

For example, if we consider two consumer FBs registered to the same event the sequential hypothesis defines that "This is interpreted as two connections $A \rightarrow B$ and $A \rightarrow C$, with event $A \rightarrow B$ emitted first and $A \rightarrow C$ the second"[15]. This is at

least an ineffective implementation. An event is issued once by the producer FB instance and the time and order of notification of consumer FB instances is defined by the way that event connections are implemented. What authors describe as "APPLICATION IN THE IEC 61499 DESIGN LOOP" [15] is just an argument from the same authors to emphasize the completely arbitrary hypotheses made by the sequential hypothesis paradigm and its usefulness. However, authors in [15] arbitrarily claim that "The sequential model is expected to be immediately applicable, implemented in a number of software tools." And also that the "parallel model also has some benefits, especially for hybrid and pure hardware implementations." disputing in this way all the benefits of the concurrent paradigm in software engineering.

It should be noted that there is a need for both FB models, i.e. the 1131 and the 61499, to support the two distinctly different approaches for the design of real-time systems, i.e. the event-triggered and the time triggered. This distinction is made based on the triggering mechanisms for the start of processing and communication activities of the real-time application. According to the event-triggered model, the processing and communication activities of the FB model should be initiated whenever a significant change of state, i.e. an event other than the regular event of a clock tick is noted. In the time-triggered approach, all activities are initiated at predetermined points in time [25].

6 Conclusions

The Function block model has a long history in the control and automation systems domain. IEC61131 is widely accepted and used in the industry. However, it implies several limitations in addressing the demands of today's complex embedded systems in this domain. The proposal of the IEC to address these limitations, i.e. the 61499 FB model, was not well accepted by the industry. It is assumed to provide solutions to the limitations of the 1131 and exploit current practices from software engineering but this is not the case. It fails in several very important objectives and does not provide a promising vehicle for industry to address the challenges of the next generation embedded industrial systems. Even more, in the case that the IEC1131 working group adopt the proposed OO extension to the 1131, as it is expected, the future of IEC61499 is questionable.

References:

- International Electrotechnical Commission. IEC International Standard IEC 61131–3:2003, Programmable Controllers, Part 3: Programming Languages, 2003.
- [2] Oded Maler, "On the programming of Industrial Computers", June 4, 1999.
- [3] PLCOpen for Effiency in Automation, http://www.plcopen.org/
- [4] R. Lewis, Modelling control system using IEC 61499: Aplying function blocks to distributed systems, The Institue of Electrical Engineering, IEE control engineering series; no. 59,2001
- [5] International Electro-technical Commission, (IEC), International Standard IEC61499, Function Blocks, Part 1 - Part 4, IEC Jan. 2005. (http://www.iec.ch/)
- [6] K. Thramboulidis, "IEC61499 Function Block Model: Facts and Fallacies", IEEE Industrial Electronics Magazine (forthcoming).
- [7] B. Heck, L. Wills, and G. Vachtevanos, "Software Technology for Implementing Reusable, Distributed Control Systems," *IEEE Control Systems Magazine*, Vol.23, Issue 1, February 2003 Page(s):21-35.
- [8] Szyperski, C., Component Technology What, Where, and How?, 25th Inter. Conf. On Software Engineering (ICSE'03).
- [9] Matthews, R.S.; Muralidhar, K.H.; Sparks, S.
 "MAP 2.1 conformance testing tools", IEEE Transactions on Software Engineering, Volume 14, Issue 3, March 1988 Page(s):363 – 374
- [10] Repenning, A.; Ioannidou, A.; Payton, M.; Wenming Ye; Roschelle, J.; Using components for rapid distributed software development, IEEE Software, Volume 18, Issue 2, March-April 2001 Page(s):38 – 45
- [11] G. Doukas, K. Thramboulidis, "A Real-Time Linux Based Framework for Model-Driven Engineering in Control and Automation" IEEE Transaction on Industrial Electronics, (forthcoming).
- [12] V. Vyatkin, V. Dubinin, Sequential Axiomatic Model for Execution of Basic Function Blocks in IEC61499, 5th IEEE Inter. Conf. on Ind. Informatics (INDIN 07), July 23-27, 2007, Vienna, Austria, Volume: 2, Page(s): 1183-1188
- [13] OMG, Unified Modeling Language: Superstructure, ver. 2.1.1, formal/2007-02-03.
- [14] Von der Beek, A comparison of statechart variants, In Formal Techniques in Real-Time and Fault-Tolerant Systems, L. de Roever and J. Vytopil, Eds. Lecture Notes in Computer Science, vol. 863, Page(s): 128–148.

- [15] V. Vyatkin, Victor Dubinin, Carlo Veber, Luca Ferrarini, Alternatives for Execution Semantics of IEC61499, 5th IEEE Inter. Conf. on Ind. Informatics (INDIN 07), July 23-27, 2007, Vienna, Austria, Vol. 2, Page(s): 1151-1156.
- [16] G. Doukas, K. Thramboulidis, "A Real-Time Linux Execution Environment for Function-Block Based Distributed Control Applications", 3nd IEEE International Conference on Industrial Informatics, Perth, Australia, August 2005, (INDIN'05), Page(s): 56-61.
- [17] K. Thramboulidis, A. Zoupas, Real-Time Java in Control and Automation: A Model Driven Development Approach, 10th IEEE Intern. Conf. on Emerging Technologies and Factory Automation, (ETFA'05), Catania, Italy, September 2005, vol.1, Page(s): 38-46.
- [18] K. Thramboulidis, "A model based approach to address inefficiencies of the IEC61499 function block model", 19th Int. Conf. on Software and Systems Engineering, Dec. 2006, Paris, Page(s): 9.
- [19] G. Booch, "Object Oriented Analysis and Design", the Benjamin/Cumming Series, second edition 1994.
- [20] K. Thramboulidis, "IEC 61499 in Factory Automation", International Conference on Industrial Electronics, Technology & Automation, (CISSE'05 - IETA), Dec. 10-20, 2005, Page(s): 115-123.
- [21] V. Kulkarni, S. Reddy, "Separation of Concerns in Model-Driven Development", IEEE Software, Vol. 20, Issue 5, Sept.-Oct. 2003 Page(s):64 – 69
- [22] Solberg, A.; Simmonds, D.; Reddy, R.; Ghosh, S.; France, R.; "Using aspect oriented techniques to support separation of concerns in model driven development", 29th Annual International Computer Software and Applications Conference, 2005. COMPSAC 2005. Volume 1, 26-28 July 2005 Page(s):121 126 Vol. 2
- [23] E.W. Dijkstra, "On the role of scientific thought". Selected writing on Computing: A Personal Perspective, Springer-Verlag, 1982.
- [24] C. Reade, Elements of Functional Programming, Addison-Wesley Longman Publishing Co., Inc., 1989.
- [25] Kopetz, H., Real-Time Systems: Design Principles for Distributed Embedded Applications, Kluwer Academic Publischers, 1997.
- [26] K. Thramboulidis, D. Perdikis, S. Kantas, "Model Driven Development of Distributed

Control Applications", The International Journal of Advanced Manufacturing Technology, Volume 33, Numbers 3-4 / June, 2007, Springer-Verlag, Page(s):233-242.

- [27] G. Cengic, O. Ljungkrantz, K. Akesson, "Formal Modeling of Function Block Applications Running in IEC 61499 Execution Runtime", 11th IEEE Intern. Conf. on Emerging Technologies and Factory Automation, Sept. 20-22, 2006, Czech Republic.
- [28] K. Thramboulidis, G. Doukas, "IEC61499 Execution Model Semantics", International Conference on Industrial Electronics, Technology & Automation, (CISSE-IETA 06), Dec. 4-14, 2006, Page(s): 223-228.
- [29] Doukas, G., K. Thramboulidis, "Implementation Model Alternatives for IEC 61499 Function Block Networks", 6th IEEE Intern. Conf. on Industrial Informatics, July 13-16, 2008, Daejon, Korea, Page(s): 295-300.
- [30] Thramboulidis, K. "Model Integrated Mechatronics – Towards a new Paradigm in the Development of Manufacturing Systems", IEEE Transactions on Industrial Informatics, vol. 1, No. 1. February 2005, Page(s): 54-61.
- [31] Strömman, M.; Thramboulidis, K.; Sierla, S.; Papakonstantinou, N.; Koskinen, K. "Incorporating Industrial Experience to IEC 61499 Based Development Methodologies and Toolsets", Proc. of the 12th IEEE Intern. Conf. on Emerging Technologies and Factory Automation (ETFA 2007), Patras, Greece, 25-28 September, 2007, Page(s): 490-497.
- [32] Daniel Witsch, Birgit Vogel-Heuser, "Close integration between UML and IEC 61131-3: New possibilities through object-oriented extensions", Proc. of the 14th IEEE Intern. Conf. on Emerging Technologies and Factory Automation (ETFA 2009), Mallorca, Spain, 22-22 September, 2009
- [33] Industry Day, 13th IEEE Intern. Conf. on Emerging Technologies and Factory Automation (ETFA 2008), 15-18 Sept, Hamburg, Germany, 2008, http://www2.hsuhh.de/aut/ETFA_2008/Industry_Day_files/ETF A%202008%20Industry%20Day.pdf