On Comparisons of the ISaGRAF implementation of IEC 61499 with FBDK and other implementations

Valeriy Vyatkin[1] *Senior Member IEEE* and Julien Chouinard[2] *Non-member*

[1] Department of Electrical and Computer Engineering,
The University of Auckland, Auckland 1142, New Zealand, E-mail: v.vyatkin@auckland.ac.nz
[2] ICS Triplex ISaGRAF *Inc*, Canada, E-mail: jchouinard@icstriplex.ca

Abstract. **This paper presents first results of comparison of ISaGRAF implementation of IEC 61499 with that of FBDK – the tool traditionally used for experiments with this standard in the research community.**

I. INTRODUCTION

The use of distributed networking controllers becomes a common place in industrial automation. IEC 61499 [1] is an architecture for distributed automation software which has raised many expectations among researchers and automation practitioners. The expectations are, partially, due to the huge success of the IEC 61131-3 standard, that defines programming languages for programmable logic controllers (PLC). IEC 61499 was perceived as a follow-up to IEC 61131-3, which added to the expectations.

For years IEC 61499 was in focus of research works. Many research groups worldwide have contributed to developing case studies and even prototypes of supporting tools. The main software tool, used in the research community in the research period was Function Block Development Kit (FBDK) [2], consisting of a development environment (FBEditor) and run-time platform (FBRT). FBDK is written in Java and compiles function blocks to Java classes, so Java is a strong pre-requisite for using FBRT on particular embedded targets. However, the use of Java, although beneficial for portability, by no means is implied by the standard itself. FBDK was developed by major industrial automation vendor Rockwell Automation and played very important role in IEC61499 evaluation, development and education, but was never intended for application "in the field" as it lacks many functional features of a modern automation workbench.

ISaGRAF is the first fully fledged automation product supporting the complete design chain. Its vendor ICS Triplex ISaGRAF actively promotes this product on the market. Being based on the well established IEC 61131 approach, this solution shows a pathway for the new standard to take up.

The arrival of first commercial implementations of IEC 61499 raises the question of establishing their compliance with the standard. One can outline the following groups of issues, which need to be examined:

1) Syntactic compliance can be established by direct comparison of the syntactic artefacts of the standard with the constructs of a particular implementation;

2) Semantic compliance can be established by comparison with the definitions of the standard, and, if they are not sufficient, with reference implementations.

3) Compliance with application design patterns and work practices. Although, implementation of IEC 61499 so far was mostly in research laboratories, a number of design patterns have been proposed and investigated, e.g. see [7], [8].

In this paper we addressing the question: "How far away stands ISaGRAF from the 'letter and spirit' of IEC 61499", thus trying to contribute to a methodology for establishing standard's compliance for current and future IEC61499 implementations. Another relevant problem (specifically in relation to ISaGRAF) addresses migration from cyclic scanned programmable logic controllers (PLC) of IEC61131-3 to the event driven controllers conforming to IEC61499. In particular, it is demonstrated, how the particular details of ISaGRAF can influence the developer. ISaGRAF is compared with FBDK in a constructive way: it is shown how some structures of the ISaGRAF implementation can be mapped to FBDK implementation and vice versa.

II. FUNCTION BLOCK SEMANTICS OF OTHER IMPLEMENTATIONS

We will accept for the reference execution model the definitions of the standard, formulated in an axiomatic form in [14]. The main features of that model are captured in the condensed form of four postulates, rather than much larger text, as follows:

1. A function block (FB) can be in one of the states 'active', 'idle' or 'pre-empted'. An FB's activation, i.e. a transition from 'idle' to 'active', can occur only as a consequence of an event at an event input of the block. For a basic FB activation means start of its ECC evaluation.
2. A single run of a basic FB cannot be pre-empted by another function block. It can only be pre-empted by its execution container (resource) in order to process input events.
3. A single run of a basic function block is instantaneous or "relatively short".
4. An event input variable of a function block clears after single ECC transition, regardless of was this event used in the evaluation or not.

Two more postulates determine scheduling of function blocks in a FB-network. For example, sequential execution model is achieved by the following postulates:

5. Output events are issued immediately after the corresponding action is completed.
6. If a function block emits several output events in one state of execution-control chart (ECC), they are emitted sequentially.

An alternative, parallel, execution model, proposed in [15], is based on the modified postulates (5) and (6) and

assumes the possibility of concurrent execution of two or more blocks, or ensures equivalence of results as if the execution was concurrent. This model is based on the fact that in IEC61499 the function blocks distributed across different resources can run in parallel.

FBDK is the best known among researchers and, in many aspects, can be regarded as a reference implementation of IEC 61499. Its FB execution model as implemented in FBRT was called NPMTR in [9]. The NPMTR model is based on the depth-first scheduling, when an emitted event immediately causes invocation of the recipient block, even if the emitting block has more events to emit. While partially compliant with postulates 1-4, it is different from the two scheduling models discussed above and cannot be regarded as a reference.

A number of trial run-time platforms, such as FORTE [17] and FUBER [16] aim at implementation of the event-driven invocation (EDI) nature of IEC 61499 function blocks (captured in the postulates 1-3). This model will be referred further in the text as EDI, and NPMTR can be regarded as a very close to EDI. The EDI model of FBs has been around for a few years and has not seen any substantial critique from the researchers, so we can assume that the FB-research community in general agrees with it. That is why in this paper we will compare execution features of ISaGRAF to EDI rather than to just FBDK/FBRT.

III. REFERENCE EXAMPLE

The 'LED Chaser' reference example provided with ISaGRAF demo kit and discussed in [6] will be used further in the paper to illustrate differences of this implementation from its interpretation in FBDK. The demo kit, as shown in **Figure 1**, consists of 3 identical devices, each having 2 buttons and four lamps.

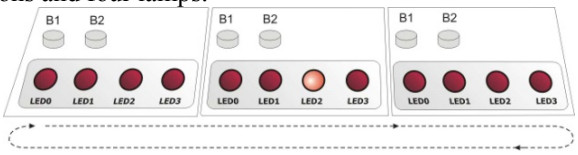

Figure 1. LED Chaser system.

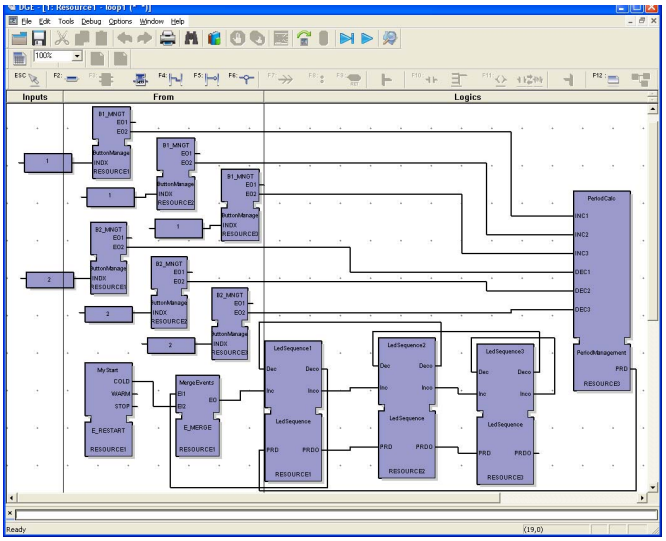

Figure 2. LedChaser FB program in ISaGRAF.

The 'LedChaser' works as follows. On the start up, the light begins 'running' from the leftmost lamp (LED0) to the right. When it reaches the rightmost lamp, the direction changes, and so on. Pressing B1 button on either device increases the speed of running, and pressing B2 decreases it. In each moment of time there is only one lit lamp in all three devices. An FB network controller for this system in the ISaGRAF version of IEC 61499 function blocks is shown in Figure 2 The control is decentralised, as pressing a button in any device will immediately affect the speed of the run even if the light is currently 'running' in another device.

For example, if a button is pressed on the device 3 while the light is "in one of the lamps" in device 1, the event will go to the PeriodManagement block, and then the new period will be transmitted to LedSequence1. As a result the speed (of light movement in device 1) will change instantaneously.

The program includes six instances of the 'ButtonManagement' FB, one per each button. This FB detects if the button is pressed, and generates event signal, which is sent to the 'PeriodCalc' FB, that generates the value of time interval between lighting of two neighbouring LEDs. This value is passed to the FBs taking care of sending signals to the lamps within one device, these are three instances of the 'LedSequence'FB type. This FB has two input events: 'Inc' and 'Dec'. When Inc is received, the FB sequentially displays all 5 possible patterns, in binary coding: '1000', '0100', '0010', '0001' and '0000', making delay of the period duration between each change of the output pattern.

a) The 'LedSequence' is a basic function block, whose logic is defined by means of execution control chart (ECC). In ISaGRAF, ECC is implemented using sequential function chart (SFC) programming language; the selection of SFC was made due to the very wide acceptance of the SFC language in the automation world and because the SFC is a state machine and hence meets the standard. However, the SFC language as per the IEC 61131 standard prescribes it would be `put on hold` after every transition. This would not have been acceptable as per the 61499 standard. Therefore the ECC in ISaGRAF, although using the syntax of the IEC61131 standard, have a bit different semantics, in accordance with the IEC61499 ECC. If evaluation of a transition between steps is ended in FALSE, the blocks execution in this scan ends, otherwise it keeps executing the ECC.

In the next scan it will be resumed at the same place with new evaluation of the transition. Consider, for example, SFC of the 'LedSequence' FB, shown in Figure 3.

In the 'LedSequence' FB this semantics allows to program output of all five sequences in one of the branches T2 – T7 (for running to the right) and T8-T13 (for running to the left). Each sequence may take considerable time (seconds) compared to the time scale of the controller (milli- or microseconds.) Thus, the function block will be entered and exited many times while a transition, say between S3 and S4, is FALSE just to check its condition.

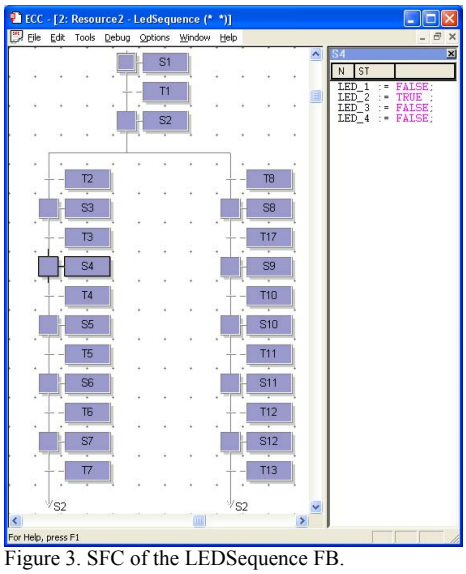

Figure 3. SFC of the LEDSequence FB.

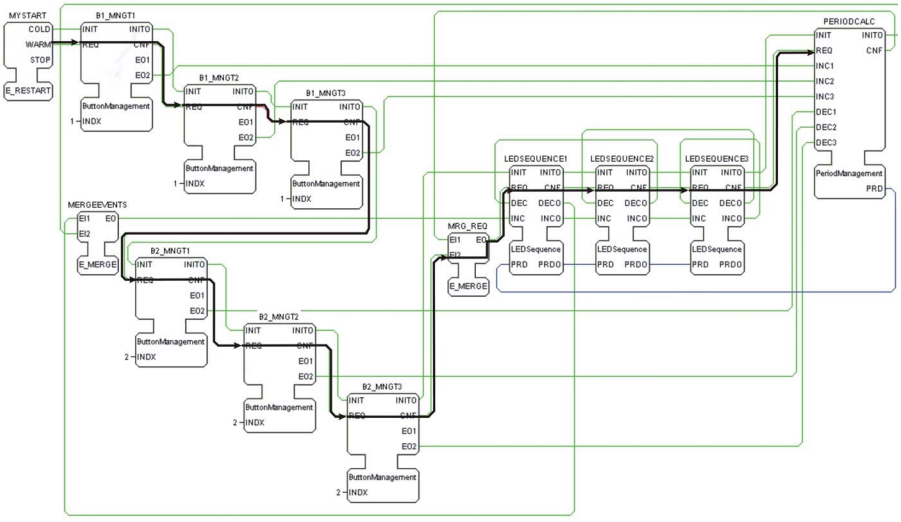

Figure 4. Chain of function blocks activation by a "PLC-like" resource.

IV. CYCLIC INVOCATION OF FUNCTION BLOCKS IN ISAGRAF

The concept of an IEC 61499 function block in ISaGRAF is based on the FB concept of IEC 61131. In ISaGRAF, within a single resource, FB ECCs are invoked in a fixed order set prior to execution. Presence or absence of events does not influence the invocation. Upon an invocation, the ECC recognizes occurrence of events. This is a bit different from the EDI approach of the standard. In ISaGRAF's resource, FB ECC's are evaluated every scan.

However, ISaGRAF offers a mechanism for using events in the block's ECCs. If an event input occurred, it can force evolution of the ECC and execution of an appropriate algorithm associated with one of the ECC actions.

Event variables in ISaGRAF are by default implemented as unsigned integers, whose increment indicates the event. This implementation is arguably more powerful since no need to stack the event. The counter can even provide "the amount of time" the event was waiting for a service.

As a result of the cyclic execution of a resource and the fact that external events and the corresponding data are read all at once by the resource at the beginning of it's cycle execution, a function block can receive several events simultaneously. This is quite different from the EDI which restricts number of "active" input events to one. The standard even does not allow more than one input event variable name in a transition condition.

Also, in ISaGRAF there is no implicit 'event clearing mechanism' in basic FBs. The programmer has full freedom in handling and interpreting events in SFC. When an event is received, the corresponding integer variable can be compared to the last event value and if the value has been incremented, the local event comparator would also be incremented thus clearing this event.

Within a single resource, a network of function blocks is evaluated sequencially following specific rules. Values of upstream function block outputs immediately become available to other downstream function blocks in the very same scan. These values otherwise (in the upstream FBs) will be used in the next scan.

With respect to the postulates 1-4 the following remarks can be made:

1. ISaGRAF is not completely complying with Postulate 1, as FB ECCs are invoked every scan. Yet, the ECC determines whether or not the algorithm should be executed. The end result is the same; even though the ECC is invoked every scan, the algorithm execution happens only with the occurrence of events. Such FBs would have minor differences from the FBs designed in the FBDK paradigm. The illustration is provided in Figure 5 by a function block detecting in which sequence the input events ei1 and ei2 arrive to the function block. The sequence <ei1,ei2> causes output event R1 and the sequence <e2,e1> - the output event R2. In Figure 5(a) the ISaGRAF version of the function block is presented, here special function calls (*LocalEventInput*) need to be executed in states S2, S3, S5 in order to detect input events. In Figure 5(b) the same function is presented as function block in FBDK paradigm. As one can see, the ECC states in the EDI solution do not include any specific actions to detect events, as the events are detected by the resource prior to FB invocation. However, the intention of ISaGRAF developers was to implement the event-driven invocation of algorithms, and this goal has been achieved.

2. ISaGRAF satisfies Postulate 2 – no function block can be pre-empted by another FB within a resource.

3. ISaGRAF satisfies Postulate 3, since its algorithms are short in execution, and no waiting is allowed within FBs.

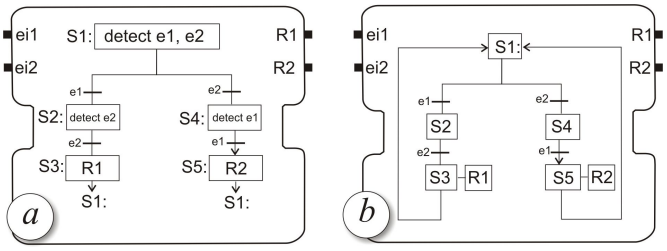

Figure 5. (a) Event sequence detector in ISaGRAF implementation and (b) in "FBDK" implementation.

This is achieved by the same means as in IEC 61131.

4. In ISaGRAF events are cleared during the process of their detection. It is different from the Postulate 4, but does not contradict to the standard, since this issue is not rigorously defined there. The event will be "ON" until the detection of that event will be next time executed, which can happen either in the next scan, or in the same scan or never happen at all. In order to implement the Postulate 4 the ECC developer needs to execute the event detection function right after the event variable was used in a transition.

A. Equivalent event-driven application

The application equivalent to the 'LedChaser', but implemented in a purely event-driven way using FBDK, is shown in **Figure 4**. The main differences with the ISaGRAF version are:

- The initialisation chain is explicitly introduced, which required all blocks to have INIT event input and INITO event output. Having such an initialisation chain is a good style proved to be beneficial in IEC 61499 programming. The INIT and INITO event signal as well as any IEC61499 signals can be implemented in the ISaGRAF ECC chart.
- To ensure that the PRD value of the PeriodManagement FB is delivered to its destination LedSequence instances, the corresponding FB type has got additional event output CNF, associated with the PRD data output. The destination FB type (LedSequence) has got an additional REQ event input and CNF event output.

As opposite to the ISaGRAF version, in this application the service interface ButtonManagement FBs ECC are not forced to execute every scan, but assumed to be activated only when the corresponding button is pressed.

The logic of the LedSequence basic FB type, however, cannot be as easily modified to suit the event-driven pattern.

One possible way to implement the SFC logic within the event-driven FB is to develop an equivalent composite FB, as it was proposed in [10]. In the following section we consider another way.

B. Explicit representation of cyclic activation using the IEC 61499 reference model

The execution model of ISaGRAF can be modelled by the EDI execution model of IEC 61499 as follows.

In the FBDK/FBRT implementation each FB needs an additional activation event input (we call it REQ). Upon execution the block needs to emit an output event (we call it CNF). All blocks within an execution container (resource) need to be connected in a cycle-free chain by event connection arcs: CNF of one block to the REQ of another, as illustrated in **Figure 4**. In ISaGRAF these event chains are not necessary (but can be implemented) as the blocks will be invoked in every cycle anyway.

In this particular case it is assumed that resources are aligned sequentially and are given an execution slot one after another, like in ISaGRAF. The chain is activated by the WARM event output of the MYSTART FB, an instance of FB type E_RESTART, which in this case is assumed to emit WARM event once upon each activation.

For basic FBs, a modification is required to represent their SFC inside an algorithm of the standard equivalent FB, as illustrated on example of the LedSequence FB in Figure 6 and is explained as follows.

An internal Boolean variable SFCACTIVE must be added to the set of internal variables. The SFC needs to be slightly modified to define explicitly its entry point and exit points. In the entry point, which is the initial step of the SFC, the variable SFCACTIVE is set to TRUE, and in the exit points it is set to FALSE.

The working is as follows. Upon occurrence of either 'Inc' or 'Dec' event inputs, the ECC jumps to the SFC state and activates the SFC algorithm. The variable SFCACTIVE becomes true in the S1 step. For example, in case of 'Inc' event, SFC evolves and stops at step S3, since transition T3 awaits the 'Period' in the step S3. At this point the SFC algorithm temporarily ends. The control is passed to the ECC, which emits the CNF event and remains in the state SFC, since no transition conditions are TRUE. The block's execution ends completely, but states of ECC and of SFC are

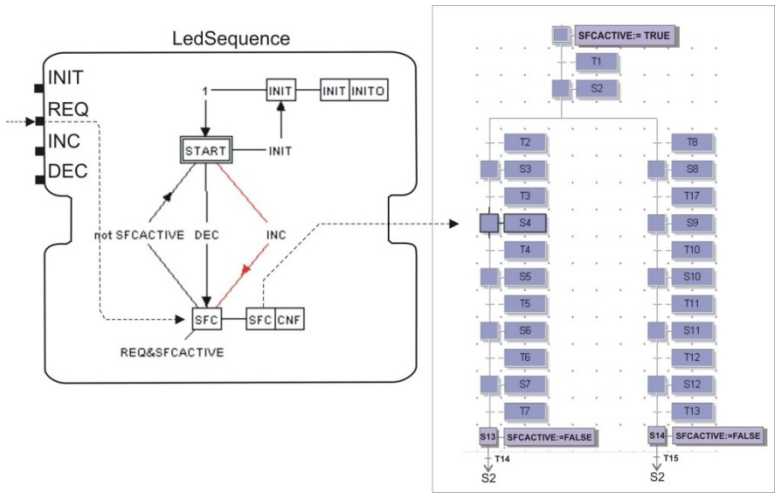

Figure 6. Basic function block (in FBDK paradigm) encapsulating SFC.

Figure 7. Distributed configuration of the LEDChaser in the FBDK paradigm.

stored.

The next time the block will be activated by event REQ (which is dedicated for activation of the function block). Due to the self-looping transition REQ & SFCACTIVE, the algorithm SFC will be entered again, and the step S3 will be active again. The process will repeat itself until the whole chain of sequences has been outputted to the lamps. The exit points are inserted after the sequence is completed, i.e. as steps S13 and S14.

One should note that implementation of this particular SFC would require a little extension of the IEC 61499 syntax, to use event variables inside algorithms. Alternatively, the INC and DEC signals would need to be implemented as data variables.

C. Distributed systems

The IEC 61499 is developed for distributed automation systems. Function blocks of one application are supposed to run in various computing devices. In our example, one can assume that each of the three LED-buttons devices has an embedded microcontroller, where some of the FBs would run. The microcontrollers need to be connected via network to pass events and data between the FBs.

In the EDI paradigm, communication is implemented in asynchronous way. Communication function blocks need to be inserted to the "cutting points" of event and data connections where they cross the resource boundaries. This is illustrated in Figure 7. Sending of events and data is implemented using PUBL FB type and the receiving by SUBL FB type. This representation with explicit communication FBs is certainly more complicated thus being more confusing for end-users.

D. ISaGRAF implementation of distribution

In ISaGRAF, the distribution is done by assigning each function block instance in the program with identifier of the resource, in which the block will be executed. Unlike FBDK, the ISaGRAF workbench does not provide a separate view for the program parts, allocated to a particular resource. That is why no communication interface function blocks are provided in the ISaGRAF standard library, since the communication is hidden from the user.

However, implementation of distributed systems inevitably requires insertion of communication functions in the positions, where event and data connection lines cross the device boundaries. ISaGRAF's compiler apparently inserts the pairs of Send and Receive functions to the code, generated for each resource and ensures that the data associates with an event are travels with the event on the message packet providing the equivalent of the WITH event-data association of the standard. Communication of ISaGRAF FB application with other IEC61499 compliant systems is implemented using specific service interface FBs, currently such an FB is provided for communication with FBs running on FBRT.

If a device has several communication interfaces, for example, Ethernet and a fieldbus, such as, Profibus, ASi or DeviceNet, it must be defined explicitly, through which of these interfaces the communication will occur. To this end, in ISaGRAF, the PLC definition file is used and imported for each configuration and thus the runtime would know on which communication network the data and event should go.

Specifically in the ISaGRAF execution model, all the 'Receive' functions of a resource are executed at the beginning of the resource activation, before executing all the FBs. All the 'Send' functions are executed at the end of each resource's activation, respectively. Such a model guarantees, that all blocks of the core application part within the resource will operate over the same input data.

V. ISAGRAF SEMANTICS FROM APPLICATION-DEVELOPER PERSPECTIVE

The major benefit expected from the event mechanism of IEC 61499 function blocks is in their improved portability and re-use. Each FB should be considered as an independent and autonomous entity, "living" in an environment that provides services enabling the FB to receive and generate events and to execute control functions with the occurrence of events. The execution order in a network of FBs shall be determined only by event flow.

The concept of "event" is crucial for implementation of this vision, and is the main difference of the IEC61499 function block concept from that of IEC61131-3. Implementation of the IEC 61499 "event" concept has been causing many sceptical comments in the past with respect to determinism of execution (such as "Events can be lost").

Even minor syntactic differences of an implementation may imply significant changes in the system behaviour or correctness. Thus, ISaGRAF's syntax does not support explicit event/data associations (WITH-connections between event and data inputs and outputs). The semantics of event/data passing between blocks in ISaGRAF assumes that an event I/O is associated to all corresponding data arcs. This assumption may result in execution logic different from the EDI model, in which the associations can be defined with more freedom.

However, it is our belief, that the most serious differences in the execution models of FB implementation of EDI and ISaGRAF may show themselves only in cases of simultaneous events being handled in one scan. Consider, for example, the FB network in **Figure 8**. Here the function block C issues output event R1 in case if ei1 arrives, R2 in case if only ei2 arrives, and R3 if both ei1 and ei2 occur

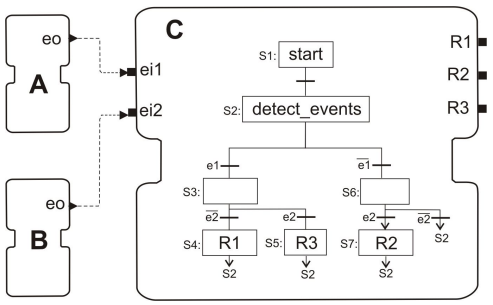

Figure 8. A function block application illustrating that the order of invocation can impact on the results of the computations.

simultaneously.

In case if both FBs A and B produce events at their outputs 'eo' and the invocation order is A,B,C or B,A,C , the result of C will be correct: R3. However, in case of the invocations in the order A, C, B or B, C, A the simultaneity of ei1 and ei2 would not be detected by C and result will be either R1 or R2.

Thus, when a network of function blocks is created, it is good to know, in which order the blocks will be invoked. In the IEC61499 context it is not always the case. Thus, mapping of application FBs to resources can be changed during system's life cycle, which may immediately impact on the behaviour.

This particular issue, however, can be quite easily fixed. Another policy of event signal delivery, based on a delayed propagation (i.e. signals generated by a function block in the current scan become available to other FBs only in the next scan), can fix this problem. Alternatively, two types of event arcs can be also introduced: one for immediate event delivery within same scan, and the other for "delayed" event delivery in the next scan.

In case of the "delayed" event delivery, the FB network in **Figure 8** will be detecting the simultaneity of ei1 and ei2 at any order of FBs invocation. However, inevitable downside of such a solution is increased number of scans, needed to produce the reaction.

It is rather a philosophical question whether function block networks can allow several events being "ON" simultaneously. It may be a good programming style to keep the event flow simple and do not allow several events to be generated/consumed by FBs within one resource/scan, but the possibility of this cannot be completely ruled out.

To avoid such problems, function blocks need to be designed so that their decision making is not relying on simultaneous arrival of several events to its inputs. As it was noted in [14], it is implicitly assumed in IEC 61499 (and in EDI) that an FB is invoked by one event, so no ECC transition condition can even include two event variables!

VI. CONCLUSION

ISaGRAF v.5.1 represents a practical solution to introduce gradually the new IEC 61499 standard and combine it with the popular IEC 61131 standard.

In our view this incremental approach has its benefits, both for the developer, who did not have to change much the tool and run-time from the IEC61331-3 version, and for the end-user, who, arguably, will easier migrate from IEC 61131-3 to IEC 61499.

There are some syntactic and semantic differences of ISaGRAF interpretation from others, which illustrate the importance of the harmonisation work currently conducted by the OOONEIDA working group on execution models.

Constant CPU load, which is the case in PLC-like control systems, like ISaGRAF, is admittedly better than sporadic load of event-driven implementations. The full load is achieved in the network thus it can not use more bandwidth. If the network accepts the bandwidth of the distributed application it always will.

The question of which semantics is better, synchronous, asynchronous, or hybrid, has not been resolved in the broader context of distributed computing. The answer largerly depends on the requirements and technical details of each particular application.

VII. REFERENCES

[1] Function blocks for industrial-process measurement and control systems - Part 1: Architecture, International Electrotechnical Commission, Geneva, 2005

[2] FBDK – Function Block Development Kit, Online: www.holobloc.com

[3] ICS Triplex ISaGRAF Workbench for IEC 61499/ 61131, v.5.1,Online:http://www.icstriplex.com/

[4] IEC 61499 Tech Notes, ICS Triplex ISaGRAF Inc, 2006

[5] Hands on IEC 61499 with ISaGRAF, ICS Triplex ISaGRAF Inc, 2007

[6] J. Chouinard et al., "An IEC 61499 configuration with 70 controllers; challenges, benefits and a discussion on technical decisions", 14th IEEE Conference ETFA'07, Patras, 2007

[7] J. Christensen, "Design patterns for systems engineering with IEC 61499," in Proc. *Verteilte Automatisierung,* Magdeburg, Germany: Otto-von-Guericke-Universität, 2000

[8] Vyatkin V., *IEC 61499 Function Blocks For Embedded and Distributed Control Systems Design*, 297p., ISA/O3neida, USA, 2007

[9] C. Sünder et al.: Usability and Interoperability of IEC 61499 based distributed automation systems, Proc. 4th IEEE Conf. INDIN06

[10] M. Riedl, C. Diedrich, F. Naumann, "SFC in IEC 61499", 13th IEEE Conference on Emerging Technologies and Factory Automation, Prague., September 20-22, 2006, pp.662-667

[11] J. Chouinard, R. Brennan, Software for Next Generation Automation and Control, 4th IEEE Intl. Conf. on Industrial Informatics, Singapore, 2006

[12] J. LM Lastra, L. Godinho, A. Lobov, R. Tuokko, "An IEC 61499 Application Generator for Scan-Based Industrial Controllers", in *Proc. of the 3rd IEEE Conference on Industrial Informatics*, Proceedings, Perth, Australia, August 2005

[13] L. Ferrarini, M. Romanò, and C. Veber, Automatic Generation of AWL Code from IEC 61499 Applications, in *Proc. of the 4th IEEE Conference on Industrial Informatics*, Singapore, August 2006

[14] V. Vyatkin, V. Dubinin, *Sequential Axiomatic Model for Execution of Basic Function Blocks in IEC61499*, 5th IEEE Conference on Industrial Informatics (INDIN'07), Proc., pp. 1183-1188, Vienna, 2007

[15] V. Vyatkin, V. Dubinin, Ferrarini, L.M., Veber C. *Alternatives for Execution Semantics of IEC61499*, 5th IEEE Conference on Industrial Informatics, Proc., pp. 1151-1156, Vienna, 2007

[16] G. Čengić, O. Ljungkrantz, and K. Åkesson, "*Formal Modeling of Function Block Applications Running in IEC 61499 Execution Runtime*," in Proc. of 11th IEEE Conf. ETFA 2006, Prague

[17] C. Sünder, A. Zoitl, J.H. Christensen, M. Colla, T. Strasser "Execution Models for the IEC 61499 elements Composite Function Block and Subapplication", In Proceedings of IEEE Int. Conference on Industrial Informatics, Vienna , 2007