

# Alternatives for Execution Semantics of IEC6149

Valeriy Vyatkin, *Senior Member, IEEE*, Victor Dubinin, *Non-member*,  
Carlo Veber, *Member, IEEE*, Luca Ferrarini *Senior Member, IEEE*

**Abstract** – This paper contributes to the ongoing development of comprehensive execution semantics of IEC61499 by discussion and comparison of the semantics that could be achieved by alteration of postulates introduced in the Sequential Hypothesis, and by variation of different priority assignments during implementation.

## I. INTRODUCTION

The current state of IEC61499 [1] allows for various execution semantics of distributed control applications. This was illustrated in a number of works, for example in [2],[3],[4]. This fact undermines the prospects of the standard's industrial acceptance and motivates the quest for a formally defined and comprehensive semantic or semantics.

Works [4]-[5] introduced an axiomatic definition of the IEC61499 semantic based on a set of 6 postulates.

In this paper we show that different semantics can be achieved by a simple alteration of the predicates. Moreover, work [6] introduced implementation idea based on scheduling of function blocks determined by priorities of certain operations. It was proposed there that different scheduling policies can be achieved by simple re-assignment of priorities. In this paper we will further illustrate this idea.

So far there have been different semantic ideas tried in research implementations. The NPMTR model (“Non-Preemptive Multi-Threaded Resource”) is implemented in FBDK/FBRT [7]. Sequential semantic was discussed in [2], [3], [4], and was implemented in run-time platforms  $\mu$ Crons and FUBER respectively. The model used in the Archimedes run-time environment (which is a part of the CORFU development framework [10]) is different from NPTMR in several features, for example, allowing independent event queues for each function block. Semantic based on PLC-like scan of inputs followed by subsequent re-evaluation of FB – network was developed in [15], [16], [18]. The essential difference of these

approaches is in the way how blocks in the network are activated which depends on the way of passing event signals between functional blocks.

## II. FUNCTION BLOCKS

The IEC61499 architecture is based on the concept of *function block*. The concept is analogous to the ideas of component, such as software component from software engineering and IP capsule used in hardware design and embedded systems. IEC61499 is a high level architecture not relying on a particular programming language, operating systems, etc. The same time it is precise enough to capture the desired function unambiguously.

The original desire of the IEC61499 developers was to encapsulate the behavior inside a function block with clear interfaces between the block and its environment. The idea is illustrated in Figure 1 (left side) on example of a function block type X2Y2\_ST computing on request  $OUT=X^2-Y^2$ . Interface of the block consists of event input REQ, data inputs X and Y, event output CNF, and data output OUT.

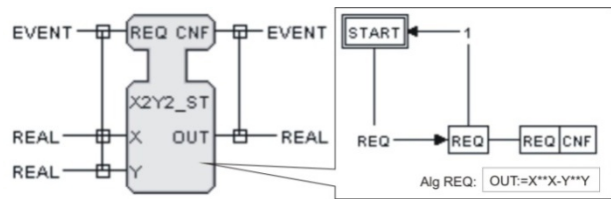


Figure 1. A basic function block type description: interface, ECC and algorithm REQ.

Note the vertical lines, one connecting REQ with X and Y, and the other connecting CNF and OUT. These lines represent association of events and data. The meaning of the association is: only those data associated with a certain event will be updated when the event arrives.

State machine is a simple visual yet mathematically rigorous way of capturing behavior. It is widely used in computer applications. In basic function blocks of IEC61499 a state machine (called Execution Control Chart, ECC for short) defines the reaction of the block on input events in a given state. The reaction can consist in execution of *algorithms* computing some values as functions of input and internal variable, followed by emitting of one or several output events. In Figure 1 the ECC and algorithm are shown in the right side. State REQ has one associated *action* that consists of an algorithm REQ

Manuscript received January 14, 2007. This work was supported in part by the research grant 3607207/9273 of the University of Auckland Research Council.

Valeriy Vyatkin is with the ECE Dept, University of Auckland, New Zealand, phone: +649-373-7599; fax: +649- 3737461; e-mail: v.vyatkin@auckland.ac.nz).

Victor Dubinin is with the University of Penza, Russia, e-mail: victor\_n\_dubinin@yahoo.com

Carlo Veber and Luca Ferrarini are with Politecnico di Milano, Italy, P.za Leonardo da Vinci 32Milano, 20133, Italy, e-mail: veber@elet.polimi.it and luca.ferrarini@polimi.it.

and emitting of output event CNF afterwards. The algorithm computes  $OUT := X^2 - Y^2$ .

Networks of function blocks are used in IEC61499 as the main enabler of distributed systems modeling. An example is given in Figure 2. Here the same  $X^2 - Y^2$  function is implemented as a network of three function blocks, doing addition, subtraction and multiplication. This network can be encapsulated in a composite function block having the same interface as X2Y2\_ST from Figure 1.

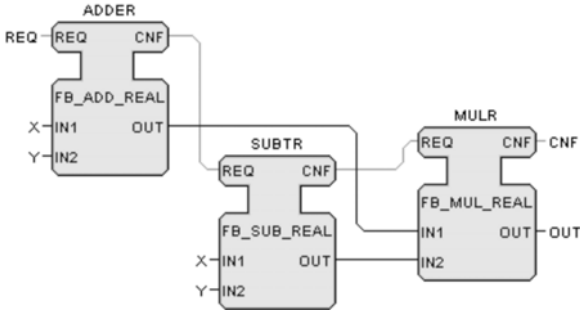


Figure 2. Implementing  $X^2 - Y^2$  as a network of function blocks.

The network could also be executed in a distributed way. The IEC61499 architecture implies two stage design process supported by the corresponding artifacts of the architecture: *applications* and *system configurations*. An application is a network of function block instances interconnected by event and data links. It completely captures the desired functionality but does not include any knowledge of the devices and their interconnections. Potentially, it can be mapped to many possible configurations of devices depending on their computational capability. A suitable automatic mechanism to find the best map between FBs and devices, in terms of memory consumption and application temporal performance, has been proposed by the European project TORERO [17]. The proposed allocation algorithm takes into account: devices memory and communication protocol, memory and time needed for a single FBs execution step, application real-time constraints (inserted in event connections and FBs algorithm execution following the TORERO specification). As finalization of such a process, the TORERO development environment automatically generates the needed communication FBs. A system configuration adds these fine details, representing the full picture of devices, connected by networks and with function blocks allocated to them.

### III. FACTS ABOUT SEMANTICS OF IEC 61499 FUNCTION BLOCKS

The Standard defines the following execution of a function block as a sequence of eight (internal) events (with timestamps  $t_1$ - $t_8$ ) illustrated in Figure 3.

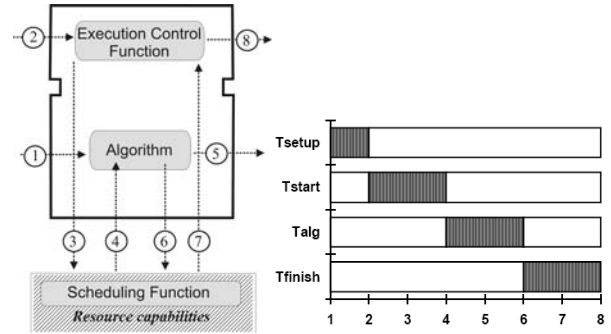


Figure 3. Sequence of events at a function block invocation and essential time intervals [1], 4.5.3

- $t_1$ : Relevant input variable values (i.e., those associated with the event input by the WITH qualifier) are made available.
- $t_2$ : The event at the event input occurs.
- $t_3$ : The execution control function notifies the resource scheduling function to schedule an algorithm for execution.
- $t_4$ : Algorithm execution begins.
- $t_5$ : The algorithm completes the establishment of values for the output variables associated with the event output by the WITH qualifier.
- $t_6$ : The resource scheduling function is notified that algorithm execution has ended.
- $t_7$ : The scheduling function invokes the execution control function.
- $t_8$ : The execution control function signals an event at the event output.

As shown in Figure 3, right side, the significant timing delays in this case which are of interest in application design are:

- $$T_{\text{setup}} = t_2 - t_1$$
- $$T_{\text{start}} = t_4 - t_2 \text{ (time from event at event input to beginning of algorithm execution)}$$
- $$T_{\text{alg}} = t_6 - t_4 \text{ (algorithm execution time)}$$
- $$T_{\text{finish}} = t_8 - t_6 \text{ (time from end of algorithm execution to event at event output)}$$

Annex A to the standard provides a number of pre-defined event processing basic function blocks and their reference implementations, and states that the parameters  $T_{\text{setup}}$ ,  $T_{\text{start}}$  and  $T_{\text{finish}}$  are considered to be zero (0) for these implementations. The parameter  $T_{\text{alg}}$  is considered to be equal to the parameter DT for E\_DELAY type, and to be zero (0) for all other component function blocks in the reference implementation.

The behaviour of a basic function block upon occurrence of an input event is described by the Table 1 in 5.2.2 whose (partial) copy is presented below in Figure 4.

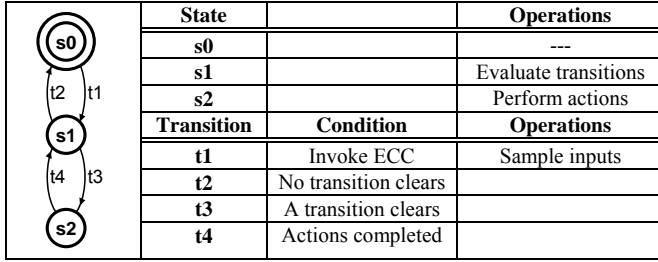


Figure 4. Table 1 from [1], 5.2.2, defining the ECC operation state machine.

#### IV. AXIOMATIC DEFINITION OF THE SEMANTICS

A few assertions regarding the execution of function blocks can form an axiomatic basis of its execution model. It would be wise to keep this set of *postulates* as small as possible. We assume that all other properties of an execution model can be derived from the postulates. Altering one or several postulates can give us completely different execution model. The following set of postulates, known as Sequential Hypothesis, was introduced in [4]:

1. A function block can be in one of the states ‘active’, ‘idle’ or ‘pre-empted’. An activation (i.e. the transition from ‘idle’ to ‘active’) can occur only as a consequence of event at an event input of the block.
2. A single run of a basic FB – an activity between transition t1 and t2 of the state machine in Table 1 – cannot be pre-empted by another function block. It can only be pre-empted by resource in order to process input events.
3. A single run of a basic function block is instantaneous or “relatively short”
4. Event input of a function block clears after single ECC transition, regardless of was this event used in the evaluation or not.
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 ECC, they are emitted sequentially.

The 6<sup>th</sup> postulate defines the behaviour exemplified in Figure 5. We believe that the spirit of the 6<sup>th</sup> postulate implies the execution model which we further refer to as “Sequential hypothesis”. Removing the 6<sup>th</sup> postulate will imply a possibility of parallel function block execution within one resource. This model may also look quite feasible and is implementable. However, we believe that it does not accord well to the letter and the spirit of the standard. Argumentation in favour of the 6<sup>th</sup> postulate is as follows.

From that one can conclude that even though output events can be emitted during a function block run, their recipient will not be started immediately, but will be just scheduled and started after this run is completed.

#### V. EMITTING OUTPUT EVENTS FROM FUNCTION BLOCKS

Let us consider example in Figure 5. FB0 is activated by event input EI and DI in that moment is TRUE. The run of FB0 will consist of STATE2 and STATE3. When the

ALG1 is completed and EO1 is to be issued, the question is if the FB1 will be activated immediately or after the run of FB0 is completed? Then, in STATE3, the question is whether to issue EO2 right after ALG2 is completed or wait until ALG3 completes and issue EO2 and EO3 together? In the latter case, shall FB2 and FB3 be started simultaneously? Answers to these questions are determined by the postulates #2, #5 and #6. Since (according to #2) FB0 cannot be pre-empted by FB1, the EO1 needs to be stored.

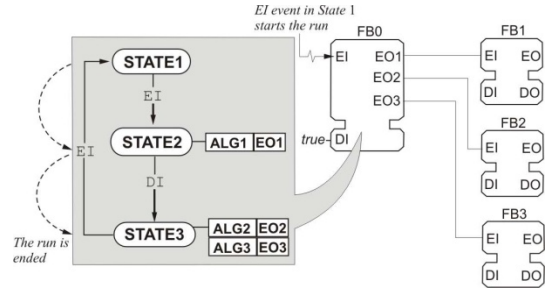


Figure 5. Run of FB0 consists of passing through STATE2 and ending in STATE3, issuing EO1, EO2 and EO3.

Although the Standard leaves unanswered the issue when the output events of a Basic FB are issued, three options are possible:

- Output event is emitted immediately after the corresponding algorithm in the action is completed;
- After all actions of a state are completed;
- After the *single run* of the basic FB is completed.

The interpretation 3) was explicitly present in the previous drafts of the standard but was discarded. According to that interpretation, events EO1, EO2 and EO3 in our example should be emitted together at the end of the run.

From that we conclude that the interpretation 3) is less attractive than 1) and 2). The interpretations 1) and 2) are not different in the order of generated events and can be only different in their timestamps.

We adopted the interpretation 1) which forms the Postulate #5.

As for the states with several actions, the Standard says explicitly that the actions associated with an ECC state are executed sequentially, one after another. Following the spirit of Postulate#5 it is quite logical to assume that in a single state output events shall be executed immediately after the corresponding algorithm completes, i.e. sequentially with respect to each other. **This forms the Postulate #6.**

Both these postulates 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.

Indeed, what happens after an output event is emitted, is defined by the clause 3 in 5.3.2 as follows:

“If an event output of a component function block is

connected to an event input of a second component function block, occurrence of an event at the event output of the first block shall cause the scheduling of an invocation of the execution control function of the second block, with an occurrence of an event at the associated event input of the second block.”

### VI. HOW TO IMPLEMENT THE SEQUENTIAL HYPOTHESIS

Implementation of the sequential hypothesis requires further details as follows:

1. Emitting an event by a function block means request to the scheduler of the resource to pass the event to the destination function block. The scheduler ensures that two or more event inputs of a basic FB or SIFB never occur simultaneously;
2. A resource (FB scheduler) maintains the queue of external events and can be in one of two states (see the state machine in Figure 6):
  - Receptive to external events in SIFBs;
  - Not receptive to external event in SIFBs (when executing FB). When an event arises in such a state it is stored in a FIFO queue. The queue is processed in the next receptive state of the resource.

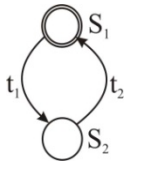
State/Transition	Description
	Check if the queue of external events is not empty <b>While</b> the queue is not empty <b>For each</b> event : Take (and remove) it from the queue; Activate the recipient SIFB and pass the events arising from its execution to the FB scheduler of the resource;
S <sub>2</sub>	FB Scheduler selects next FB from its queue and starts its execution
t <sub>1</sub>	Queue of scheduled blocks is not empty
t <sub>2</sub>	Scheduled block completed its execution

Figure 6 State machine and algorithm of resource operation.

Let us consider an example: a network of function blocks shown in Figure 7. The network is allocated to a device that has single resource.

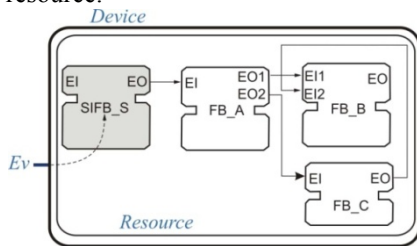


Figure 7. Function block network activated by an external event through block SIFB\_S

The device has one input channel which is polled by the function block SIFB\_S. Arrival of an external event at this channel causes activation of the rest of the network. The execution process is illustrated in Figure 8.

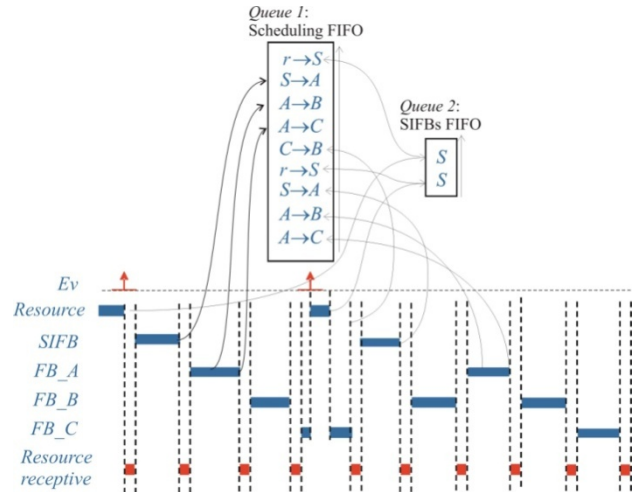


Figure 8. Timing diagram representing activity of blocks in the network.

Scheduling of block activations and registration of arrived external events is implemented using two queues of First-In First-Out kind (FIFO): Scheduling FIFO, further referred to as *Queue 1* and SIFBs FIFO, referred to as *Queue 2*. Queue 1 stores the events passed from a block to another block. Queue 2 stores occurrences of external events that occur while the resource is in the non receptive state S<sub>2</sub>. It is needed to ensure the processing of input events will follow the order of their occurrence. In our example, the network is activated by arrival of an external event to the input channel *Ev*. Arrival of an external event interrupts the execution of a function block which is preempted by execution of the resource, which places the corresponding request to the Queue 2. This can be seen in case of the execution of FB\_C. Pre-emption of the FB\_C execution by the Scheduler after the second occurrence of *Ev* does not contradict the 3<sup>rd</sup> postulate as no other FB is executed while the pre-emption occurred, so the integrity of the FB network evaluation is not affected.

When resource returns to the receptive state it checks if the Queue 2 is not empty and adds the request of kind  $r \rightarrow S$  to the Queue 1, clearing the Queue 2. Then it takes next request from the top of Queue 1 and executes the corresponding function block (that is target of the request, i.e. in case of request  $r \rightarrow S$  the service interface block *S* is executed.)

**Remark:** In a more general case the queues 1 and 2 may not necessarily be FIFO but just sets of requests where scheduler places the requests and from where it selects an event to activate next function block.

### VII. IS SEQUENTIAL SEMANTIC THE BEST FOR DEVELOPERS?

Benefits of the Sequential Hypothesis for application developer are:

- more predictable model of execution in terms of

reaction time and invocation sequence;

- requires less resources i.e. will lead to potentially more efficient implementations;
- direct call invocation in NPMTR model requires stack space and context switching;
- can be implemented in one thread;
- allows customisation for handling real-time constraints, and other priority mechanisms;

The question of “How good is this model for developers” can be answered only after certain trial period of using the tools and devices that comply with this semantic model. The most popular at the moment FBDK/FBRT follows different semantic (NPMTR), so the final comparison is yet to be made. However, there is a possibility of an alternative semantic which could allow for parallel execution of function blocks.

As illustrated in Figure 9, splitting an event connection to two or more blocks (left side), is currently interpreted as implicit E\_SPLIT block (right side), that leads to sequential scheduling of FB2 and FB3 under Sequential Hypothesis.

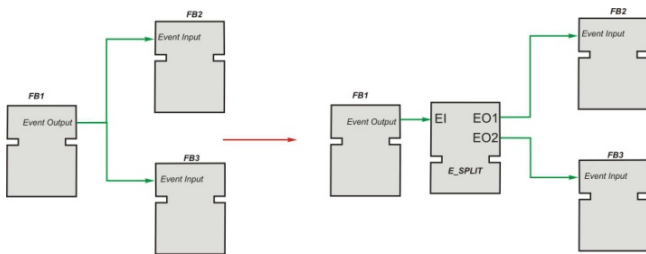


Figure 9. Event fork can lead to parallel execution of function blocks.

However, the event fork in Figure 9 can be interpreted by many developers as parallel invocation of FB2 and FB3, and this vision is quite intuitive. The main reason to think about sequential execution of function blocks is the sequential nature of modern computers. The function blocks are not necessarily to be implemented as sequential software code, they could be compiled into hardware form, where it is only natural to work in parallel.

The parallel execution of function blocks within a resource can be achieved by modifying Postulates #5 and #6, for example as follows:

**Postulate #5'** Output events are issued immediately after the corresponding action is completed. The destination function block is invoked right after the event is issued.

**Postulate #6'** If a function block emits several output events in one state of ECC, they are emitted simultaneously.

With this set of postulates our execution model becomes Parallel Hypothesis, whose discussion, however is beyond the scope of this paper. We acknowledge that there could be good reasons in favour of Parallel Hypothesis, such as:

- better fit to the ‘not so short’ algorithms assumption;
- more intuitive in ‘event-forking’ situations;
- better fit to multi core hardware implementation;

To find out more justification for appropriate function block scheduling strategies, we need to consider the origins of distributed system configurations in the design loop.

#### VIII. APPLICATION IN THE IEC 61499 DESIGN LOOP

An application is a network of function block instances. An example is shown in Figure 10. The application consists of four function blocks named A, B, C and D. For simplicity we connected the blocks with only event connectors, omitting data exchange. Note the connection from A is split on two: to B and to C. This is interpreted as two connections A→B and A→C, with event A→B emitted first and A→C the second. According to the standard, execution of each block takes a *short* time interval.

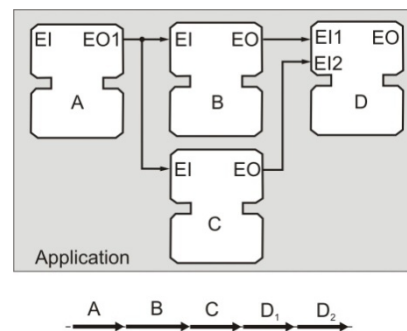


Figure 10. An example of *application* and the execution order following the sequential scheduling idea

Entire application can be deployed on a single device. If the device implements a sequential scheduling mechanism then the sequence of activations will be as shown in the Figure. (We denoted  $D_1$  and  $D_2$  invocations of the block D by event inputs EI1 and EI2 respectively).

The application design is the first part of system development. The next step is its deployment that is a mapping of function blocks to *devices*. Same application can be mapped onto one device, or on two or more devices in arbitrary combinations as illustrated in Figure 10 and in Figure 11, a) and b) respectively.

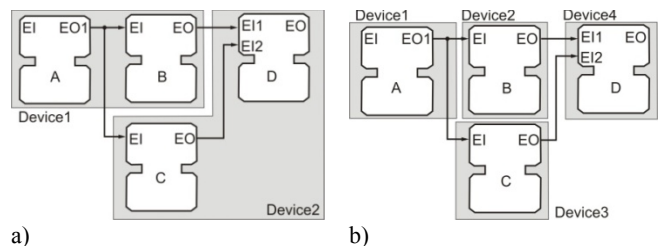


Figure 11. Two (out of many) possibilities to map an application on several distributed devices

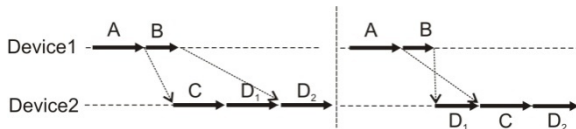


Figure 12 Sequence of function block execution in mapping to two devices (different sequences are due to asynchrony of devices and of communication between them).

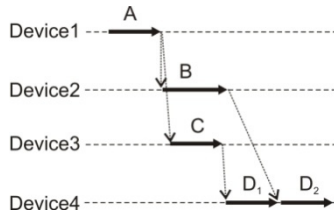


Figure 13. Sequence of function block execution in mapping to and to four devices (C).

In distributed configurations some of the blocks can be **executed concurrently**. This can lead to different sequences of output events for the same input conditions if the devices are not synchronized. Properties of communication channels can also influence sequence of output events. This is illustrated in Figure 12, a) and b) for the two-device configuration from Figure 11 a). The event connection  $A \rightarrow C$  needs to be implemented as event passing through a communication channel between devices 1 and 2. If the communication protocol does not preserve sequence of events and the channel can introduce arbitrary delays, than the execution sequence can differ drastically (compare Figure 12, sequences a) and b)). The original application model interpreted according to the sequential hypothesis would eliminate these execution sequences. In case of the four-device distribution there would be even more variants, one of which is shown in Figure 12, c).

From the above said, one can conclude that the two-stage design paradigm of the IEC 61499 implies that application must have execution semantics that would not change after mapping to any configuration of devices. In this case it can be used for simulation after the first stage of the design and formal analysis.

A network of function blocks can adequately represent a (potentially) distributed system only if there was a mechanism of concurrent activation of blocks. This requires ‘tweaking’ of two mechanisms: 1) simultaneous appearance of events at several outputs of a single function block; 2) event propagation enabling simultaneous delivery of events to inputs of several blocks.

## IX. CONCLUSIONS

This paper presented an overview of motivations for the definition of formal execution semantics of IEC61499. Sequential and parallel execution models are discussed. The sequential model is expected to be immediately applicable, implemented in a number of software tools. The parallel model also has some benefits, especially for hybrid and

pure hardware implementations.

## X. REFERENCES

- [1] Function blocks for industrial-process measurement and control systems - Part 1: Architecture, International Electrotechnical Commission, Geneva, 2005
- [2] Zoitl A., Grabmair G., Auinger F., and Sunder C. Executing real-time constrained control applications modelled in IEC 61499 with respect to dynamic reconfiguration, 3<sup>rd</sup> IEEE Conference on Industrial Informatics, Proceedings, Perth, Australia, August 2005
- [3] G. Čengić, O. Ljungkrantz, and K. Åkesson, “Formal Modeling of Function Block Applications Running in IEC 61499 Execution Runtime,” in 11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Prague, September 2006.
- [4] V. Vyatkin, V. Dubinin, Execution Model of IEC61499 Function Blocks based on Sequential Hypothesis, paper draft, [http://www.ece.auckland.ac.nz/~vyatkin/o3fb/vd\\_seqsem.pdf](http://www.ece.auckland.ac.nz/~vyatkin/o3fb/vd_seqsem.pdf)
- [5] V. Vyatkin, V. Dubinin, Sequential Axiomatic Model for Execution of Basic Function Blocks in IEC61499, INDIN, 2007
- [6] V. Dubinin, V. Vyatkin, On Definition of a Formal Semantic Model for IEC 61499 Function Blocks, Journal of Embedded Systems, 2007, submitted
- [7] Function Block Development Kit (FBDK), <http://www.holobloc.com/doc/fbdk/index.htm>
- [8] OONEIDA Workgroup on Execution Semantic of IEC61499: [http://www.ooneida.org/standards\\_development\\_Compliance\\_Profile.html](http://www.ooneida.org/standards_development_Compliance_Profile.html)
- [9] C. Sünder, A.Zoitl, J. H. Christensen, V. Vyatkin, R. Brennan, A. Valentini, L. Ferrarini, K. Thramboulidis, T. Strasser, J. L.Martinez-Lastra, and F. Auinger: Usability and Interoperability of IEC 61499 based distributed automation systems, 4<sup>th</sup> IEEE Conference on Industrial Informatics (INDIN 2006), Proceedings, Singapore, 2006
- [10] Thramboulidis K.S. “Using UML in Control and Automation: A Model Driven Approach”, 2<sup>nd</sup> international Conference on Industrial Informatics INDIN’04, 24-26 June 2004, Berlin, Germany
- [11] L. Ferrarini and C. Veber, Implementation approaches for the execution model of IEC 61499 applications, 2<sup>nd</sup> IEEE Conference on Industrial Informatics, Proceedings, Berlin, June 2004
- [12] V. Dubinin, V. Vyatkin “Formalized definition and modelling of IEC 61499 function block systems”, Letters of Tertiary Education Institutions, Volga region, Russia, Penza State University Publishers, 2005, N 5, pp.76-89
- [13] V. Dubinin, V. Vyatkin, Towards a Formal Semantics of IEC 61499 Function Blocks, 4<sup>th</sup> IEEE Conference on Industrial Informatics (INDIN’2006), Singapore, 2006
- [14] V. Dubinin, V. Vyatkin, H.-M. Hanisch, “Using Prolog for Modelling and Verification of IEC 61499 Function Blocks and Applications”, 11th IEEE Conference on Emerging Technologies and Factory Automation (ETFA 2006), Proceedings, Prague, 2006
- [15] L. Ferrarini, M. Romanò, and C. Veber, Automatic Generation of AWL Code from IEC 61499 Applications, 4<sup>th</sup> IEEE Conference on Industrial Informatics (INDIN 2006), Proceedings, Singapore, 2006
- [16] L. Ferrarini, C. Veber, F. Ferrari, G. Fogliazza, Applied Metamodelling to convert IEC61499 Applications into Step7 environment, ECC07, European Control Conference 2007, 2-5 luglio 2007, Kos, Greece, accepted.
- [17] L. Ferrarini, C. Veber, Control function design and implementation of distributed automation systems for manufacturing applications, IJMR, International Journal of Manufacturing Research, Vol. 1, No. 4, Inderscience Enterprises Ltd, 2006, pp. 442-465
- [18] J. LM Lastra, L. Godinho, A. Lobov, R. Tuokko, An IEC 61499 Application generator for Scan-Based Industrial Controllers, 3<sup>rd</sup> IEEE Conference on Industrial Informatics, Proceedings, Perth, Australia, August 2005