Design and Implementation of LabVIEW-Based IEC61499 Compliant Device

Grzegorz Polaków and Mieczyslaw Metzger

Department of Automatic Control, Electronics and Computer Science
Silesian University of Technology, ul. Akademicka 16, 44-100 Gliwice, Poland
{grzegorz.polakow,mieczyslaw.metzger}@polsl.pl

Abstract. The method of IEC 61499 compliant device implementation with the National Instruments LabVIEW is proposed. The work focuses on these aspects of the tasks of the event generation and dispatching, which have no direct counterparts in the G language. A mapping of all the IEC 61499 concepts onto the G language concepts is described. Because of the limited multithreading support in LV (multithreading is possible but a number of parallel threads is fixed at the stage of the program development and compilation) it is needed to fit all the IEC 61499 defined functionalities in a fixed number of threads. A FIFO queue based dispatching algorithm is implemented, similar to the one used in the C++FBRT implementation. The ultimate objective of the work is the development of the FBLV run-time environment, which converts the LabVIEW compatible industry grade hardware into the IEC 61499 compliant device.

Keywords: distributed control, IEC 61499, holonic systems, LabVIEW, run-time environment.

1 Introduction

The first standard for holonic industrial systems is the IEC 61499 [1]. It took place of previously existing non-standardized products, however the market still lacks any successful industrial implementations. The most known run-time environment for the IEC 61499 is the FBRT developed by Holobloc [2], which is a part of the FBDK. The FBDK framework is designed primarily as a reference environment [3] as it exploits every aspect of the standard. The FBDK is developed using the Java programming language. There are not many industry-grade hardware implementing the Java Virtual Machine, which limits possible industrial use of the FBRT software. The process of industrial adaptation of the IEC 61499 advanced when the ICS Triplex [4] included the support for the standard in their ISaGRAF software suite. It allows to develop event-based applications, which may be targeted for the large set of supported operating systems. Drawbacks of this solution are: the lack of support for the purely industrial hardware, and controversial method of algorithm scheduling using the scan-based approach [5]. There are few more run-time environments implementing the norm compliant events and function blocks (see [6] for a brief survey), but usually those environments rely heavily on an underlying operating system. The only embedded environment known at the moment is the C++ conversion of the FBRT [7][8], which apparently did not get beyond the stage of research.

V. Mařík, T. Strasser, and A. Zoitl (Eds.): HoloMAS 2009, LNAI 5696, pp. 183–192, 2009.
© Springer-Verlag Berlin Heidelberg 2009

This paper presents the progress of the development of a run-time environment complying to the IEC 61499 norm (see [9] and [10] for reference). The environment is developed using the G language of the LabVIEW platform [11]. The G language is equipped with a wide set of capabilities, some of which are similar to the concepts of the norm (i.e. event queues, basic OOP). However, the G language constructs are general in nature, and using them to implement the norm compliant behavior requires much work. Proposed mapping of the concepts of the IEC 61499 norm onto G language constructs is the main contribution of this work. With the approach presented in this paper it is possible to program the LabVIEW compatible industry-grade hardware in the norm compliant way. The ultimate objective of the work described is the development of complete run-time environment (called from now on as a FBLV) which, when uploaded to a PAC (Programmable Automation Controller – see [11]), converts the PAC into the IEC 61499 compliant device.

It should be noted that the FBLV is not designed as a complete framework exploiting fully all the concepts of the standard. The goal of the implementation is to develop a run-time environment for industry-grade hardware, because the standard did not receive any attention from the biggest automation equipment manufacturers, which resulted in a lack of IEC 61499 compliant equipment (for now the most popular solution of this problem was uploading the FBRT to the Netmaster family controllers).

At the moment, preferred method of the FBLV device programming is by uploading an XML file generated by the FBDK, describing the function block network. However, in future it is expected that the FBLV device will be fully cooperative online as a part of system configuration. Such run-time environment would largely expand the set of the IEC 61499 compliant hardware by inclusion of all the PLCs and PACs made by National Instruments. An additional contribution of the work is the description of proposed single threading scheduling algorithm and low-level implementation of the norm defined FBs. This description can be easily adopted for other programming languages with limited multi threading capability.

2 General Concept

The G language used by the National Instruments LabVIEW development environment is a graphical language exploiting the concept of the function blocks similar to the FBD language of the IEC 61131 standard [12]. The G language was developed at first as a tool simplifying the data acquisition process, but nowadays it is more general. LabVIEW implements the concepts of loops, data structures, subprograms, GUIs, etc. – all of them realized in the form of dataflow driven block diagrams. The language is compilation-based, the diagrams are compiled into executable code, which may be targeted for various platforms, including FPGAs and PACs. The code is generated with a support for multithreading, parallel fragments of code are executed as separate threads. The most obvious drawback of the G is the complete lack of C-style memory pointers, which makes it impossible to construct, for example, linked list. The other problem is the limitation of the dataflow-based multithreading. While parallel loops are executed as concurrent threads, the number of threads is determined automatically (depending on the structure of the code and the capabilities of the target hardware) and fixed at the compilation stage. In effect, LV lacks the mechanism

similar to the Java thread class – it is not possible to execute a given subprogram as a separate thread. However, LabVIEW is aware of the event-based programming concept. Typically, events are used in LabVIEW for GUI servicing, but it is possible to define user's own class of events. Such user-created classes of events are then dispatched with the LabVIEW built-in event manager in a G language construct called the Event Structure.

2.1 Mapping the Basic Concepts of IEC 61499 onto G Language

The standard defines three classes of devices, numbered from 0 to 2. The FBLV was designed to implement the functionality of the class 1 device. The class 0 was considered as too simple – it is more fitted for the intelligent sensors or actuators. For effective programming of the control algorithms, the support for FB instances is required, which leaves classes 1 and 2 under consideration. However, class 2 device, while more powerful, requires a compilation or an interpretation of a FB block algorithm given by an user. A run time-compilation of LabVIEW subprograms is not possible, a LabVIEW application has to be compiled and uploaded to the embedded device at once. An interpretation of the FB source code during run-time could be implemented, but it would lack the performance required in industrial use. In effect, the FBLV was decided to implement the functionality of class 1 device. The class 1 is well balanced between simplicity and capabilities, assuming that the library of FB types provided by the device is rich enough.

To keep the prototype implementation simple, it was decided that the FBLV will not provide a support for the multiple resources. The FBLV device accommodates a network of function blocks directly, which is equivalent to zero resources [9].

Events and scheduling algorithm. It is proposed to implement the events defined by the standard directly as the LabVIEW events serviced with the Event Structure. In consequence, the IEC 61499 events which are issued in the network of function blocks are stored in the LabVIEW internal event queue, which is working according to the FIFO principle. The IEC 61499 events are defined as the LabVIEW user events (distinct to the LabVIEW standard GUI events) carrying additional data – reference number of the block instance and the event input number at which the event arrived. Using the data carried by the event, the scheduling algorithm thread determines the proper reaction to the event, as shown in the Fig. 1.

The consecutive steps of the event dispatching process are as follows (numbered as in the Fig. 1):

1. The event is removed from the queue. Using the variables carried by the event, it is determined on which input of which of the existing FB instances the event is issued.
2. Using the table of the existing FB instances the FB type of the considered FB instance is read. Knowing the state of the FB instance (stored in the table of instances), the FB type, and the considered event input, the Event Dispatcher determines a chunk of code to be executed. The algorithm performs the task defined by IEC 61499 for the given input of a given FB type.

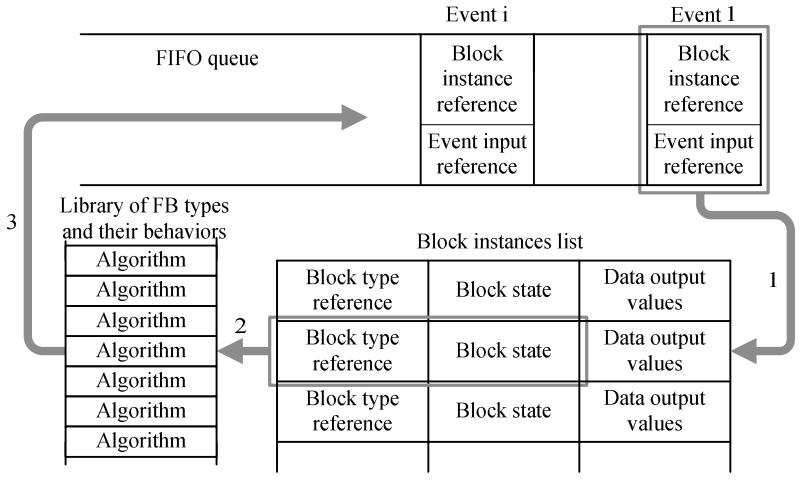

Fig. 1. The idea of the scheduling algorithm

3. If the task is supposed to result in issuing the new event, the proper entry is added to the queue of events waiting to be dispatched (after resolving the FB network and finding the destination FB instance and input).

A similar scheduling algorithm introduced in [6] by Zoitl et al. was named the Event Dispatcher. Due to this similarity, the scheduling algorithm proposed for the FBLV platform will be called the same. The work principle of the Event Dispatcher algorithm has a peculiar property: all the events issued in the FB network are serviced in the same processing thread, the events are dispatched sequentially. In effect, each of the events is eventually dispatched, events cannot be lost, as it is observable in the scan-based approach [13].

Block types. In the most of the existing implementations, the FB types are represented as the classes of a object oriented programming language (e.g. in FBDK). Algorithms executed by the events incoming to FBs are implemented as the methods of the classes. Although the LabVIEW implements the basics of the OOP, it is limited by the nature of the data-flow based programming, i.e. it lacks the dynamic thread creation. In effect, implementation of the FBs as the LabVIEW classes is unreasonable. Instead, a procedural programming based concept is introduced. Each of the FB type definitions is divided into basic procedures implementing the FB reaction to the events at the specific inputs. These procedures are spread and programmed as parts of the Event Dispatcher code. If a function block uses the ECC and its behavior depends on its internal state, a procedure additionally reads the state of the FB and executes a proper subroutine.

This distribution of the FB types behavior amongst the code of Event Dispatcher is the most important modification of the usual approach, resulting in the possibility of the FBLV development. The Event Dispatcher is simply the collection of basic algorithms, which are executed in an order defined by a FB connection network. It should be noted that the internal FB algorithms do not have to be developed exactly as the

norm proposes – only the external behavior of the given algorithm has to be exactly as the standard defines. ECCs defined by the standard do not need to be implemented to the letter. ECCs are, in the case of the FBLV, less the programming tool, more a guideline for the programmers of the Event Dispatcher algorithms.

The algorithm collection stored in the Event Dispatcher is described by the global memory structure describing the implemented FB types, their names, inputs and outputs. This structure is the connection between the naming introduced by the standard and the pieces of code distributed in the Event Dispatcher.

Block instances. In the OOP-based implementations, FB instances of a given FB type are the objects of the given class. In the FBLV FB instances are stored as entries in the separate memory structure. The structure contains two fields describing properties of the instance: number of the FB type of which it is an instance, and the *variant* structure (similar to the non-typed memory pointer in C language) storing the state of the FB instance. The state consist of current values of data outputs of the instance, and any data specific to the instance (file handles, network socket handles, number of internal state according to the ECC, etc.). The creation of the FB instance consists of simple adding of the entry in the table and initiating its state.

Event and data connections. The connections between the FB instances are stored in an additional memory structure. A record of the structure contains four fields: the reference to the source FB and its connected output number, and the reference to the destination FB and its input number. There are two separate structures for the event connections and for the data connections. The structure of event connections is used directly by the Event Dispatcher, so it can find the correct destinations of newly induced events (step 3 in the Fig. 1). The structure of data connections is used by the algorithms, which find the source FB instance of the data values they require, so the correct entry containing the values can be found in the instances table.

Data types. The large part of the data types defined by the IEC 61131 standard (from where, they were adopted to the IEC 61499) have native equivalents in the G language. Integer and floating typed are currently implemented in the FBLV using the native types. STRING and WSTRING are easily implementable using the LabVIEW built-in string type. The IEC 61499 specific data types i.e. color, arrays and matrices have native equivalents. The time and date, user-defined and complex types are the most difficult, the proper implementation method is not determined at the moment. The TIME type bears the specific problem, as its resolution (i.e. 1μs) is out of LabVIEW capabilities – see Timed Events section below.

The most effective way of all the data types implementation is the development of the classes hierarchy, similar to the FBRT. The G language limited OOP support is sufficient enough for such an application.

2.2 Sources of Events

The presented idea of the Event Dispatcher assumes that all the events issued in the FB network are produced by the algorithms called during dispatching the previously existing events. Function blocks which are the sources of the events do not fall into this category, and they have to be implemented in other way. The two categories of such FBs are: the service interface blocks, and the blocks issuing the output events

according to the pass of time. The solutions proposed below are basically a LabVIEW specific implementation of timed interrupts, which are the most fit for this task and used in the C++FBRT [14].

Timed events. The standard defines three FBs measuring the time, i.e. E_DELAY, E_CYCLE and E_TRAIN. Because the latter two may be implemented using the E_DELAY, it is actually the only time related FB requiring implementation. In the FBLV it is proposed to implement the second event dispatcher specifically for this task. The Timed Events Dispatcher is not based on a FIFO queue as the original one, instead its buffer holds the unordered list of the events which are scheduled to be issued by E_DELAY blocks in a future. A separate thread (i.e. parallel *while* loop) cyclically checks the buffer with a minimal achievable in LV period, removes the events due in the last period, and adds them to the main Event Dispatcher. The LabVIEW specific problem is the lack of widely compatible precise hard real-time timers. The only real-time loop provided by the G language, which is compatible with all the National Instruments hardware, has the resolution of 1ms. More precise timers are available on selected platforms only. At the current stage of FBLV development, for the compatibility, it is assumed that the 1ms resolution is enough, but it surely may be not precise enough in industrial applications. As a consequence part of the TIME data type capacity is ignored.

Service interface blocks. A similar idea has to be implemented for service interface blocks. Blocks which send information and do not expect reception confirmation may be treated as any other blocks, and implemented with the main Event Dispatcher. However, reception of incoming data which results in issuing new events has to be serviced in a separate software thread. Therefore, opening of the network sockets is done within the main Event Dispatcher, handles to the sockets are stored as the state variable in the table of FB instances. A separate thread cyclically checks opened listening sockets. If any data arrived and is stored in the buffers of network adapters, the thread reads the data, assigns it to the outputs of a proper FB instance, creates the IND event, and adds it to the main event queue.

3 Implementation

The following sections present few fragments of already developed source code, which may serve as the reference for the developers of similar solutions. The code presented is developed with the G language, but it is described thoroughly, so the general idea is adoptable in the other programming languages.

3.1 Event Dispatcher and Function Block Algorithms

The source diagram of the Event Dispatcher thread is shown in the Fig. 2. It is relatively simple, because it uses the LabVIEW built-in Event Structure. At first, the class of user events is created (1), which carries two variables: the FB block instance number and the event input number. The event class is registered (2), so the events of the class may be dispatched in the Event Structure (3) placed in the infinite While Loop (4). The diagram also illustrates the concept of FB type specific algorithms. Using the FB Instance number (5) the subVI (6) finds the FB type number of the instance in the

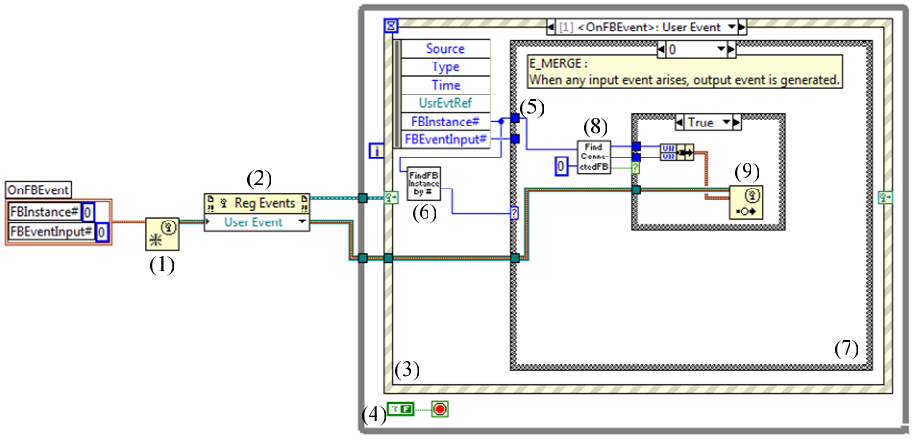

Fig. 2. The source diagram of the scheduling algorithm and the E_MERGE algorithm

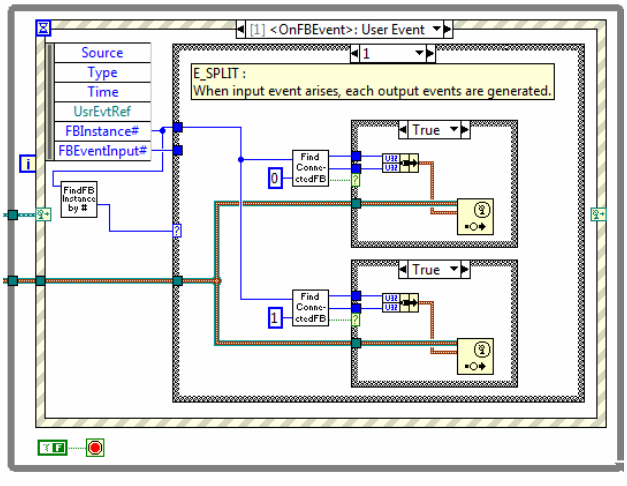

Fig. 3. The E_SPLIT algorithm

instances table. The FB type number selects one of the cases from the Case Structure (7). In the Fig. 1 the implementation of E_MERGE FB type is shown. If the FB type number equals zero (in the FBLV zero is assigned to the E_MERGE type) the diagram fragment shown in figure is executed. The algorithms issues the output event regardless of the input event number which induced the algorithm. At first, the search for the FB instance connected to the output of the serviced FB instance is performed (8). If found, the event at its proper event input is added to the Event Dispatcher queue (9).

For comparison, in the Fig. 3 the internal algorithm of the E_SPLIT FB type is shown. When an event arises at the input of the E_SPLIT instance, a search for the instances connected to both the event outputs of the E_SPLIT instance are performed. Events at inputs of those FB instances are then added to the event queue. All the blocks defined by the standard are implemented in a similar way in the Event Dispatcher code.

The algorithms embedded in the Event Dispatcher are supplemented by the global array of clusters (a cluster is a LV counterpart of a C structure) describing all the implemented FB types and their inputs and outputs. The cluster is shown in the graphical form in the Fig. 4. The same figure shows also the instances table. In the figure three FB instances are declared, two of the E_MERGE type and one of the E_SPLIT type.

Additionally, there also exist arrays of clusters not included in the figures, i.e. the table of event and data connections (containing the pairs of FB instances numbers and event/data input/output numbers), and the table of defined data types.

At the current stage of the implementation progress, the only method of interaction with the run-time environment is the GUI pictured in the Fig. 5. FB instances and connections are presented in the textual form. FBs and connections can be created and deleted. To help in the debugging tasks, the history of the events fired is provided. In

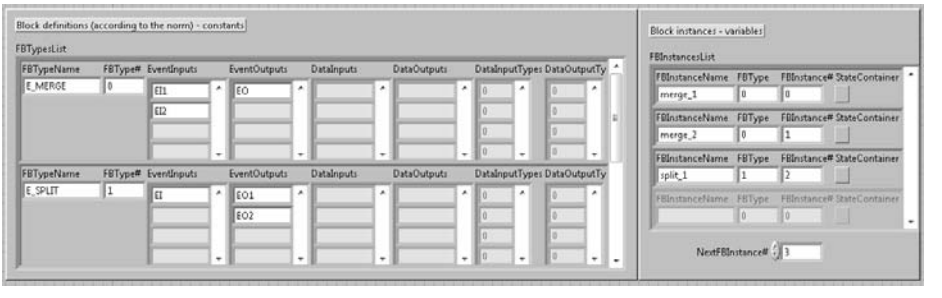

Fig. 4. The description of implemented FB types and the list of currently existing FB instances

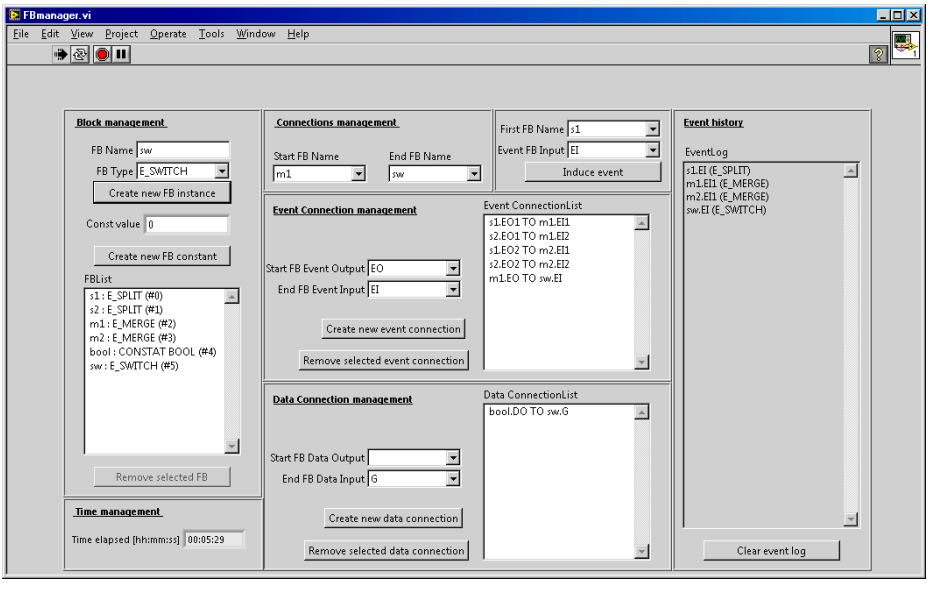

Fig. 5. The user interface of the current pilot implementation of the FBLV

further development a support for XML files is planned, so the files generated by the FBDK could be parsed into FBLV specific FB network description. It should be noted that the stage of the FB network creation and the stage of Event Dispatcher work are not separated. Both the threads work parallel, enabling the support for the dynamic reconfiguration of the system. It is possible due to the complete lack of structure compilation – all the connections between the blocks are stored and resolved with the use of dynamic memory structures.

4 Concluding Remarks and Future Work

In this work the method of IEC 61499 compliant device implementation with the National Instruments LabVIEW is described. The work focuses on those of the tasks of the event generation and dispatching, which have no direct counterparts in the G language. Because of the limited multithreading support in LV (multithreading is possible but a number of threads is fixed at the stage of the program development) it is needed to fit all the IEC 61499 defined functionalities in a fixed number of threads. Currently the FBLV run-time environment is at the development stage, but since all the main concepts are formulated the full functionality is achievable. The current prototype implementation uses desktop PCs as a hardware platform, and the interaction with the outer world is HMI based. In future, it is planned to change the target platform to the PAC hardware, which will effectively enable the creation of industrial grade IEC 61499 compatible run-time environment. The most challenging task is the implementation of timed events with the 1μs resolution, which will require the use of hardware clocks existing in the PAC platforms.

It should be noted that the performance and reliability of the FBLV approach are still not verified experimentally. The comparison of the FBLV with the other IEC61499 approaches is planned as soon, as the FBLV development advances enough.

The approach based on the dynamic memory structures and FIFO queue enables the FBLV to change the structure of the FB network without stopping the main Event Dispatcher thread. In effect, the FBLV is capable of a dynamic reconfiguration, implementation of the RECFB FB type proposed by [15] is considered in the future.

Acknowledgments. This work was supported by the Polish Ministry of Science and Higher Education using funds for 2008-2010 under grant no. N N514 296335.

References

1. IEC, Geneva. IEC 61499-1: Function Blocks – Part 1 Architecture (2005)
2. HOLOBLOC Inc.: HOLOBLOC Inc. Webpage, `http://www.holobloc.com`
3. Hall, K.H., Staron, R.J., Zoitl, A.: Challenges to Industry Adoption of the IEC 61499 Standard on Event-based Function Blocks. In: 5th IEEE International Conference on Industrial Informatics, vol. 2, pp. 823–828. IEEE Press, New York (2007)
4. ICS Triplex: ISaGRAF Webpage, `http://www.ics.triplex.com`

5. Vyatkin, V., Chouinard, J.: On Comparisons of the ISaGRAF implementation of IEC 61499 with FBDK and other implementations. In: 6th IEEE Conference on Industrial Informatics, pp. 289–294. IEEE Press, New York (2008)
6. Zoitl, A., Strasser, T., Hall, K., Staron, R., Sünder, C., Favre-Bulle, B.: The past, present, and future of IEC 61499. In: Mařík, V., Vyatkin, V., Colombo, A.W. (eds.) HoloMAS 2007. LNCS (LNAI), vol. 4659, pp. 1–14. Springer, Heidelberg (2007)
7. Zoitl, A.: Development of an IEC 61499 based embedded control platform and integration in a distributed automation system. Master's thesis, Vienna University of Technology (October 2002)
8. Rumpl, W.E., Auinger, F., Dutzler, C., Zoitl, A.: Platforms for Scalable Flexible Automation Considering the Concepts of IEC 61499. In: Mařík, V., Camarinha-Matos, L.M., Afsarmanesh, H. (eds.) IFIP Conference Proceedings, vol. 229, pp. 237–246. Kluwer B.V., Deventer (2002)
9. Vyatkin, V.: IEC 61499 Function blocks for embedded and distributed control systems design. ISA, Research Triangle Park (2007)
10. Lewis, R.: Modelling control systems using IEC 61499. Applying function blocks to distributed systems. IEEE, London (2001)
11. National Instruments Website, `http://www.ni.com`
12. IEC, Geneva. IEC 61131 Programmable controllers – Part 3: Programming languages (1993)
13. Cengic, G., Ljungkrantz, O., Akesson, K.: Formal Modeling of Function Block Applications Running in IEC 61499 Execution Runtime. In: Proceedings of the 11th IEEE Conference on Emerging Technologies and Factory Automation, ETFA 2006, Praque (2006)
14. Sünder, C., Rofner, H., Vyatkin, V., Favre-Bulle, B.: Formal description of an IEC 61499 runtime environment with real-time constraints. In: 5th IEEE International Conference on Industrial Informatics, vol. 2, pp. 853–859. IEEE Press, New York (2007)
15. Rooker, M.N., Sünder, C., Strasser, T., Zoitl, A., Hummer, O., Ebenhofer, G.: Zero Downtime Reconfiguration of Distributed Automation systems: The εCEDAC Approach. In: Mařík, V., Vyatkin, V., Colombo, A.W. (eds.) HoloMAS 2007. LNCS (LNAI), vol. 4659, pp. 326–337. Springer, Heidelberg (2007)