Journal of Mathematical Sciences: Advances and Applications
Volume 57, 2019, Pages 15-38
Available at http://scientificadvances.co.in
DOI: http://dx.doi.org/10.18642/jmsaa_7100122051

ON THE FORMAL MODEL FOR IEC 61499 COMPOSITE FUNCTION BLOCKS

KYONGJIN SOK[1], WONJE KIM[2], SONGIL CHA[3] and KINAM SIN[4]

[1]Institute of Information Technology
University of Sciences
Pyongyang
Democratic People's Republic of Korea

[2]Institute of Information Science & Technology
National Academy of Science
Pyongyang
Democratic People's Republic of Korea

[3]Department of Computer Science
University of Sciences
Pyongyang
Democratic People's Republic of Korea

[4]Faculty of Mathematics
Kim Il Sung University
Pyongyang
Democratic People's Republic of Korea
e-mail: 15bf12025@hit.edu.cn

2010 Mathematics Subject Classification: 68M14, 68Q45.
Keywords and phrases: IEC 61499, distributed control, modelling, composite function block, function block network, software development tool.
Received April 26, 2019; Revised May 14, 2019

Abstract

The applications for IEC 61499 that is standard architecture for developing the applications of distributed control and measurement in factory automation have the connected structure of the graphical elements called BFB (basic function block), SIFB (service interface function block), and CFB (composite function block). The research on the composite function block has been regarded as important issues in implementing hierarchy, multi-functionality and simplicity of software. Nowadays many researchers have been investigated IEC 61499 in the fields of the software modelling composed of basic function block and service interface function block, the transformation from IEC 61131 to IEC 61499 and syntactic extension of ECC of the basic function block. However, work related to the mathematical modelling for IEC 61499 composite function block using in designing software with hierarchical structure is still lacking. This paper presents the mathematical model for the structure and execution analysis of IEC 61499 composite function blocks by using the notation of the set theory. Also, a subapplication configuration algorithm is suggested for the subapplication corresponding to the composite function block. Then its effectiveness through the computation experiment of several distributed control applications is shown. The proposed model can be used effectively as a basis for analyzing a runtime environment of a software tool for designing and developing the applications.

1. Introduction

IEC 61499 is an international standard which emerged at 2005 year and acknowledged officially and begins to apply for designing and building distributed control system [1-3]. For designing distributed control systems, this standard can be used to unify the design and implementation of an overall control system from field level to management. It also shows intuitively the configuration and behaviour of hardware and software using function blocks. There are described rules that are necessary for designing and implementing the distributed control system, including architectures and function blocks, software tools.

Nowadays, the IEC 61499 standard has been founded widely application in many areas of factory automation. In [4], Kashyap et al. proposed the analysis and simulation results of the performance of fault

location and isolation (FLI) using the IEC 61499, distribution automation standard in an automated power distribution feeder. In [5], Zhabelova et al. proposed the hybrid agent architecture specific to the power system automation domain using the industrial standards IEC 61850 and IEC 61499. In [6], Lindgren et al. proposed the method to provide safe end-to-end response times for distributed IEC 61499 applications communicating over switched Ethernet networks. In [7], Pinto et al. presented the results implementing the traffic light systems and PID controllers using IEC 61499 in ICARU-FB environment. A standard-based control approach for distributed energy resources is introduced and implemented [8]. The function block technique, i.e., IEC 61499, is used for the development of energy demand models as it brings advantages such as modularity, encapsulation, extensibility, and reusability [9].

The IEC 61499 standard defines execution processing of each function block and simple scheduling function on a network of the function blocks that compose of basic and service interface function blocks. Hence, the different implementations of the standard have made different assumptions about how to execute the applications. As a result, the same application might behave differently when executed on different platforms [10]. In [11], Panjaitan and Frey proposed the method for modelling of IEC 61499 functionality design using UML diagram. In [12], Fletcher and Brennan described how function blocks can be used to build the holonic manufacturing systems. In [13], Sunder et al. described the different possibilities concerning composite function blocks in detail and explained in a case study of a very simple example function block. In [14], Kim and Cha proposed the structural model of IEC 61499 application with composite functional blocks.

In [15, 16], Shaw et al. proposed new design framework with the hierarchical and concurrent novel extension of ECC (Execution Control Chart) for basic function blocks. Dai et al. [17] proposed a new methodology of migration from IEC 61131-3 to IEC 61499 function

blocks. In [18], a formal definition of an IEC 61499 application using the set theory notation is presented along with a semantic function block model that is based on a state-transition approach. This model is done by choosing the next transition from the enabled function block transitions according to the execution semantics modelled. Only sequential hypothesis execution semantic is mentioned in particular. In [19], Čengić and Åkesson presented formal definitions of the application model using definitions of the types and instances of function blocks, application state space, external input and output sets. Only semantics of basic and service interface function blocks are of importance. In [20], Čengić and Åkesson proposed the formal definitions on the three different execution models, based on the formal model described in [19], buffered sequential execution model (BSEM), non-preempted multithreaded resource (NPTMR), cyclic buffered sequential model (CBEM), and shows its comparison results. Carlson and Lednicki [21] presents a simple model of the applications with different function blocks, and analyzes its runtime behaviour. Other attempts at formal modelling and verification of control related languages have been published ([22], [23]). Sequential function charts (part of IEC 61131) are modelled and verified by using time automata [22] while in Beauvais et al. [23] state charts are formally modelled by using the SIGNAL language.

The previous researches of the application model are described mainly on the formal model for IEC 61499 applications that compose of basic and service interface function blocks and are not discussed profoundly for the formal definitions and execution analysis on composite function blocks that can represent simply and hierarchically the application with complex structure and behaviour.

This paper presents the formal definitions of the composite functional blocks of the IEC 61499 standard. The mathematical definitions have been used as a basis for the implementation of a software tool for a runtime environment and formal verification. This paper is organized as

follows. Section 2 describes the structure of the function blocks in the IEC 61499 standard. Section 3 presents formal definitions on the composite function blocks of IEC 61499 standard. Section 4 shows the result of computational experiments for IEC 61499 applications by using proposed models. Finally, Section 5 concludes this work.

2. Structure of Function Blocks in IEC 61499

The software architecture defined by the IEC 61499 standard is based on functional software units called function blocks that include own algorithms and internal variables. There are three types of function blocks: basic function blocks, composite function blocks, and service interface function blocks. A basic function block executes an elemental control function, such as reading a sensor or setting the state of an actuator. The different function blocks may be combined together to encapsulate a higher-level control function, and such a combination is called a composite function block. The service interface function block provides communication services among devices.

Figure 1 shows a structure of the components of a basic function block. The model distinguishes between events, data, and algorithms. The upper quadrant of Figure 1 shows the event stream, which executed the function block code. The arrival of an event executes one or more function block algorithms. After the execution of the algorithm, the function block will enable an output event. Output events are sent to other function blocks in the application and thus input events of relevant function blocks occur. The data flow is shown in the lower quadrant of the function block in Figure 1.

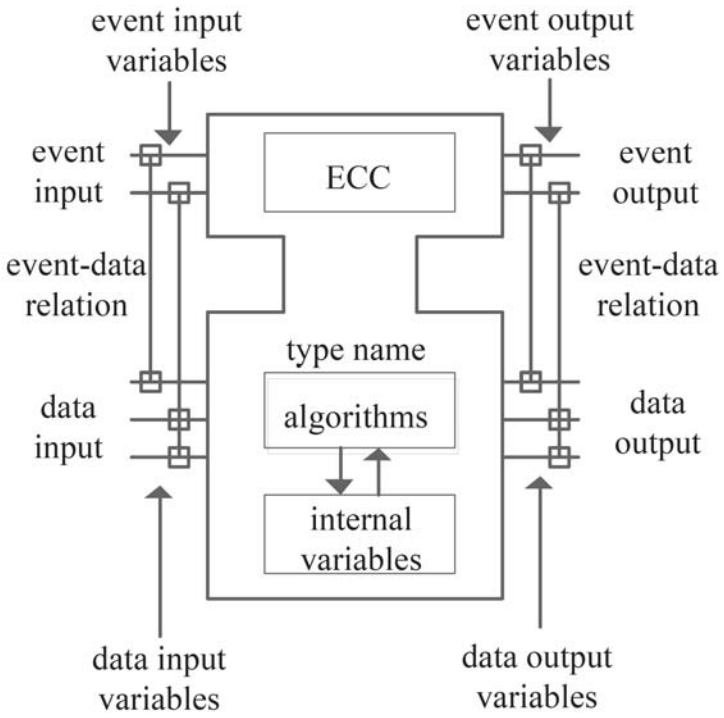

Figure 1. Components of a basic function block.

Data inputs are provided either from physical devices or from other function blocks. At the time that an input event occurs, the relevant data input values are read and the appropriate algorithm is executed using the current data values. This results in a calculation or validation that yields data outputs of a function block. When the output event is triggered, the relevant data output is made available to another function block that continues the execution of the application. Type name of the basic function block is an identifier that it can be used to uniquely identify the basic function block. The execution control chart (ECC) of a basic function block determines which algorithm to execute based on the current input event and values of input, output and internal data variables. The example ECC in Figure 2 states that if it is in the initial state, STATE0, and input event EI is received, the ECC transfers to state

STATE1 and schedules the algorithm named Algorithm for execution. After Algorithm has terminated, the output event EO emerges, and the ECC transmits immediately to state STATE0 since the transition condition is “1” (true) and finishes execution.

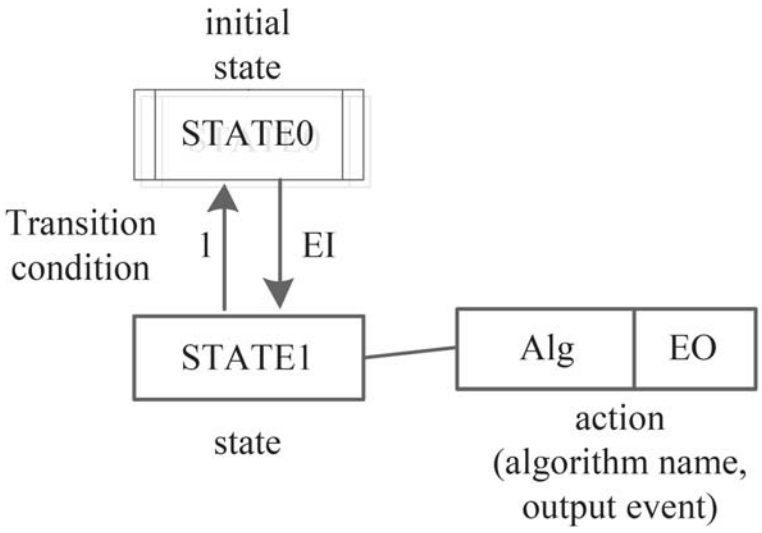

Figure 2. Example of an execution control chart (ECC).

Another types of function blocks - a composite function block has its own interface (the input and output elements for events and data) as well as a basic function block and composes of a set of several basic and service interface, composite function blocks in which function blocks are connected by events and data. A composite function block that its elements of the interface, i.e., input and output elements of events and data is connected with inner function blocks, can be used to perform higher-level control function by making subapplication with inner function blocks. A service interface function block is used to provide an application with access to services provided by an IEC 61499 execution resource, such as access to a communication network and the process under control.

We have briefly introduced the IEC 61499 standard so far. The IEC 61499 applications can be executed by using an IEC 61499 compliant runtime environment. The runtime environment may implement an IEC 61499 device composing of IEC 61499 execution resources. An important part of an execution resource is the scheduler. A scheduler's task is to provide each function block with an opportunity to execute when the block receives an event, i.e., a scheduler decides the block execution order. The following section presents the formal definitions on the composite function block of an IEC 61499 standard.

3. The Formal Definition of the Composite Function Block

This section presents how a formal model on composite function block followed an IEC 61499 standard, is defined. In [19], it was presented the formal definitions for IEC 61499 applications that compose of basic and service interface function blocks but not discussed on composite function blocks with hierarchical structure and behaviour. This section presents formal definitions on the interface, type and instance of the composite function block, and on subapplication and its execution corresponding the composite function block. Also, the composite function block that calculates function, $f(x, y) = x^2 - y^2$, is used as an illustrative example. Composite function block composes of the three basic function blocks that do the addition, subtraction, and multiplication operation, respectively. The structure of composite function blocks and of subapplication corresponding to it shows in Figure 3 and Figure 4.

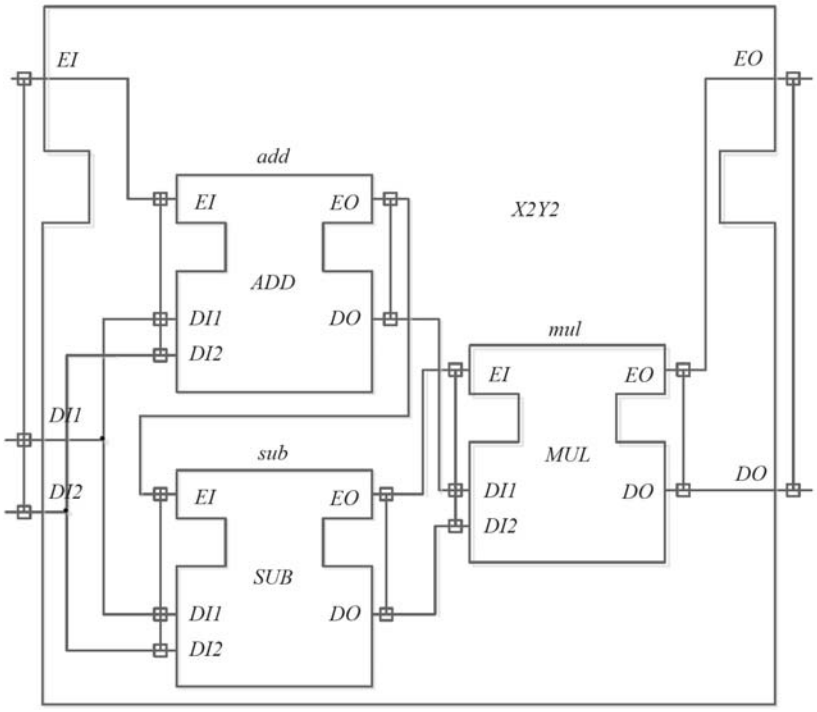

Figure 3. Structure of the composite function block $X2Y2$.

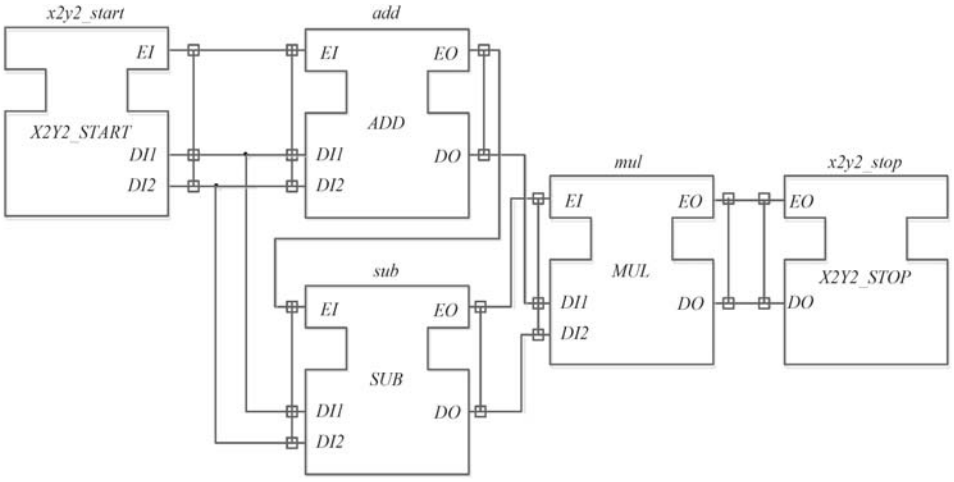

Figure 4. Structure of subapplication corresponding to a composite function block.

3.1. Structure of composite function block

First, we define the interface common to all function blocks.

Definition 1 (Function block interface)**.** A function block interface is a 6-tuple defined as

$$g = \langle E_i, D_i, \omega_i, E_o, D_o, \omega_o \rangle,$$

where E_i, D_i, E_o, and D_o are finite sets of input events, input data, output events, and outputs data, respectively. $\omega_i \subseteq E_i \times D_i$ and $\omega_o \subseteq E_o \times D_o$ are the sets of input and output associations. In the composite function block of Figure 3, interface g is represented as follows.

$$g = \langle \{EI\}, \{EO\}, \tag{1}$$

$$\{\langle DI1, * \rangle, \langle DI2, * \rangle\}, \{\langle DO, * \rangle\}, \tag{2}$$

$$\{EI \rightarrow \langle DI1, * \rangle, \langle DI2, * \rangle\}, \{EO \rightarrow \langle DO, * \rangle\}\rangle. \tag{3}$$

Line (1) describes the sets of input and output events of the block and line (2)- all values of input and output data, line (3) – the sets of input and output associations. Where symbol * indicates the range of value of data input/output variables (for example, range of value of integer type variable).

Definition 2 (Composite function block type)**.**

A composite function block type, *cfbt*, is a 3-tuple defined as

$$cfbt = \langle g, T, I \rangle,$$

where g is a composite function block interface, T is a set of types of function blocks which a composite function block composes of them, I is a set of instances of function blocks which a composite function block composes of them. A set of inner function block types and instances that are components of composite function block type, $X2Y2$, in Figure 3, is represented as follows, respectively.

$$T = \{ADD, SUB, MUL\}, \tag{4}$$

$$I = \{add, sub, mul\}. \tag{5}$$

Composite function block instance is defined next.

Definition 3 (Composite function block instance)**.**

Given composite function block type, $cfbt = \langle g, T, I \rangle$, with a composite function block interface, $g = \langle E_i, D_i, \omega_i, E_o, D_o, \omega_o \rangle$, instance, *cfbi*, of composite function block type, *cfbt*, is a 5-tuple defined as

$$cfbi = \langle n, c_e, c_d, d_i, d_o \rangle,$$

where n is instance name, c_e is a set of output event connections, c_d is a set of input data connections, d_i is an initial value set of input data variables $\left(d_i \in 2^{D_i}\right)$, d_o is an initial value set of data output variables $\left(d_o \in 2^{D_o}\right)$. The instance, $x2y2$, of composite function block type, $X2Y2$, in Figure 3, is represented as follows.

$$x2y2 = \langle x2y2, \varphi, \varphi, \{\langle DI1, 0\rangle, \langle DI2, 0\rangle\}, \{\langle DO, 0\rangle\}\rangle. \tag{6}$$

In the above representation, the sets of output event and input data connections are represented as a symbol φ (empty set), and those are determined when composite function block instance is connected with others.

3.2. Subapplication configuration algorithm

The IEC 61499 applications [19] are defined as a network of function blocks that are connected with various types of function blocks, and since composite function block inner has a network of function blocks that are connected with various types of function blocks, we can regard it as an application. In order to distinguish application that composes of inner function blocks of the composite function block, from IEC 61499 applications, we will be called it subapplication.

Events and data of composite function block interface are connected with composite function block inner function blocks. Therefore, in order to define subapplication corresponding to composite function block, we must be defined the sets of type, instance, external input and external output events for service interface function blocks of start and stop that indicates start and end blocks of subapplication. Since only service interface function block type composes of function block type interface, interfaces of start and stop service interface function block types are defined as follows.

Definition 4 (Interface of start and stop service interface function block)**.**

Given composite function block interface, $g = \langle E_i, D_i, \omega_i, E_o, D_o, \omega_o \rangle$, respectively, interfaces of service interface function blocks for start and stop are a 6-tuple defined as

$$g^{START} = \langle \varphi, \varphi, \varphi, E_i, D_i, \omega_i \rangle,$$

$$g^{STOP} = \langle E_o, D_o, \omega_o, \varphi, \varphi, \varphi \rangle.$$

Service interface function block instances for start and stop are defined such as composite function block interface. Next, we add type and instance of service function blocks for start and stop to sets of types and instances of subapplication corresponding to the composite function block. Therefore, set of type of subapplication composes of inner function block types, service interface function block types for start and stop, and set of instance composes of inner function block instances, service interface function block instances for start and stop.

Definition 5 (Sets of external input and external output events)**.**

Given type and instance of service function blocks for start and stop of subapplication, sets of external input and external output events of subapplication are defined as

$$I_e = \left\{ \langle start, e \rangle \middle| e \in E_o^{start} \right\},$$

$$O_e = \left\{ \langle stop, e \rangle \middle| e \in E_i^{stop} \right\},$$

where *start* and *stop* are instances of service function block types for start and stop.

A set of external input/output events is defined by service interface function block instances and the members in a set of external input events compose of a pair of instance name, its output event and values of related output data variables, and the members in a set of external output events compose of a pair of instance name, its input event and values of related input data variables.

Next, application state space that indicates the execution state of subapplication corresponding to composite function block, is defined as follows.

Definition 6 (Application state space)**.**

When is given a set of function block types, $T(= T_b \cup T_c \cup T_s)$, and a set of function block instances, $I(= I_b \cup I_c \cup I_s)$, of subapplication, application state space, S_a, is defined as

$$S_a = 2^{S_s \cup S_b \cup S_c},$$

$$S_s = \bigcup_{k \in I_s} \left(\left\{ n^k \right\} \times 2^{D_i^k} \times 2^{D_o^k} \right),$$

$$S_b = \bigcup_{h \in I_b} \left(\left\{ n^k \right\} \times 2^{D_i^h} \times 2^{D_o^h} \times 2^{L^h} \times Q^h \right),$$

$$S_c = \bigcup_{j \in I_s} \left(\left\{ n^j \right\} \times 2^{D_i^j} \times 2^{D_o^j} \right),$$

where radix s, b, c denote service interface, basic, and composite function block, respectively, and n is function block instance name, D_i is a value set of data input variables of function block, D_o is a value set of data output variables of function block, L is a value set of internal variables of basic function block, Q is initial state of basic function block.

The members of application state space represent all the states that all function block instances of the application may have during execution. At any time, the state of basic function block instance is represented as a pair of instance name and current values of data input/output variables and internal variables, state name, and the state of service interface and composite function block instance is represented as a pair of instance name and current values of data input/output variables. The initial state in application state space composes of initial values of data input/output variables of each function block instances when is initialized to execute the application. The initial state of application state space corresponding to subapplication of Figure 4 is represented as follows:

$$S_a^{X2Y2} = \{\langle x2y2\_start, \varphi, \{\langle DI1, 0\rangle, \langle DI2, 0\rangle\}\rangle, \tag{7}$$

$$\langle add, \{\langle DI1, 0\rangle, \langle DI2, 0\rangle\}, \{\langle DO, 0\rangle\}, \varphi, q_0\rangle, \tag{8}$$

$$\langle sub, \{\langle DI1, 0\rangle, \langle DI2, 0\rangle\}, \{\langle DO, 0\rangle\}, \varphi, p_0\rangle, \tag{9}$$

$$\langle mul, \{\langle DI1, 0\rangle, \langle DI2, 0\rangle\}, \{\langle DO, 0\rangle\}, \varphi, r_0\rangle, \tag{10}$$

$$\langle x2y2\_stop, \{\langle DO, 0\rangle\}, \varphi\rangle\}, \tag{11}$$

where q_0, p_0, r_0 are initial states of basic function blocks, *add*, *sub*, and *mul*.

The subapplication configuration algorithm corresponding to the composite function block is as follows:

Algorithm: Subapplication configuration

Input: Composite function block type $cfbt = \langle g, T, I \rangle$

Output: Subapplication, $a^c = \left\langle T^c, I^c, S_a^c, I_e^c, O_e^c \right\rangle$, corresponding to composite function block

1: $g^{START} \leftarrow \langle \varphi, \varphi, \varphi, E_i, D_i, \omega_i \rangle$ from g. // Get start SIFB type, $t^{START} = g^{START}$

2: $g^{STOP} \leftarrow \langle E_o, D_o, \omega_o, \varphi, \varphi, \varphi \rangle$ from g. // Get stop SIFB type, $t^{STOP} = g^{STOP}$

3: Get the instances of start and stop SIFB types, I^{start} and I^{stop}, respectively.

4: $T^c \leftarrow T \cup \left\{ t^{START} \right\} \cup \left\{ t^{STOP} \right\}$ // Get a set of types

5: $I^c \leftarrow I \cup \left\{ I^{start} \right\} \cup \left\{ I^{stop} \right\}$ // Get a set of instances

6: Get I_e^c and O_e^c for all SIFB instances of T^c.

7: Determine the initial states of application state space, S_a^c.

8: Returns the subapplication, $a^c = \left\langle T^c, I^c, S_a^c, I_e^c, O_e^c \right\rangle$.

3.3. Execution of composite function block

The event handling function of the composite function block instance that is used when the composite function block is executed is defined next.

Definition 7 (Event handling function of the composite function block instance)**.**

Event handling function of the composite function block instance, c^k, is calculated by the following algorithm:

Algorithm: $c^k : \langle s, t \rangle \rightarrow \left\langle s^+, t^+ \right\rangle$

1: $\langle e, s, t \rangle \leftarrow e^k \langle s, t \rangle$ // Select the event of instance

2: $\left\langle I_e^k, s, t \right\rangle \leftarrow \langle e, s, t \rangle$ // Get a set of external inputs of composite function block instance

3: execute a subapplication, $a^k = \left\langle T^k, I^k, S_a^k, I_e^k, O_e^k \right\rangle$, corresponding to composite function block instance, k.

4: *for all* $e_o^k \in O_e^k$

5: *if* $ie_o^k \neq \varepsilon$ *then* // If external output exists

6: $\langle s, t \rangle \leftarrow o^k \left\langle e_o^k, s, t \right\rangle$ // event sending function of the instance

7: *end if*

8: *end for*

9: $\left\langle s^+, t^+ \right\rangle \leftarrow \langle s, t \rangle$

where e^k is event selection function, e_o^k is an output event of composite function block instance, k, o_e^k is a set of event outputs of composite function block instance, k, o^k is event sending function.

Event handling function of the composite function block instance, when it received input event, gets external input values of composite function block instance from input event and executes subapplication

corresponding to composite function block instance, triggers output events of composite function block instance corresponding to external output values of the subapplication. Application step function that is called in application execution function [16] during the execution of subapplication corresponding to composite function block is defined in detail next.

Definition 8 (Application step function)**.**

Application step function, p, is calculated by the following algorithm:

Algorithm: $p : \langle s, t\rangle \to \left\langle s^+, t^+\right\rangle$

1: $\langle k, s, t\rangle \leftarrow i\langle s, t\rangle$ // Select the instance

2: *if* $k \neq \varepsilon$ and $type(k) \neq$' *SIFB*' *then*

3: *if* $type(k) \neq$' *CFB*' *then* // If instance is composite function block

4: $\langle s, t\rangle \leftarrow c^k\langle s, t\rangle$ // Do event handling function of the composite function block

5: else // If instance is basic function block

6: $\langle s, t\rangle \leftarrow h^k\langle s, t\rangle$ // Do event handling function of the basic function block

7: *end if*

8: *end if*

9: $\left\langle s^+, t^+\right\rangle \leftarrow \langle s, t\rangle$

where i is instance selection function, *type* is a function that returns type of function block instance, c^k is an event handling function of the composite function block, h^k is an event handling function of the basic function block. Application step function is called at each repeat processing phase of application execution function and does event

handling of the basic or composite function block instances, doesn't event handling of the composite function block. Next section shows computational experiment results for IEC 61499 applications with composite function block.

4. Experiment Results

This section shows first the results compared with previous models [16, 17] and next shows the experiment results tested in our distributed control program development tool, IIDesigner 1.0. The test was run on a Windows PC with in Intel Core i3 CPU 2.4GHz processor and 4GB RAM by the method that measures and compares the execution time during 1 million cycles on the different applications. The motor operation example by the previous and proposed model using IIDesigner is shown in Figure 5 and Figure 6.

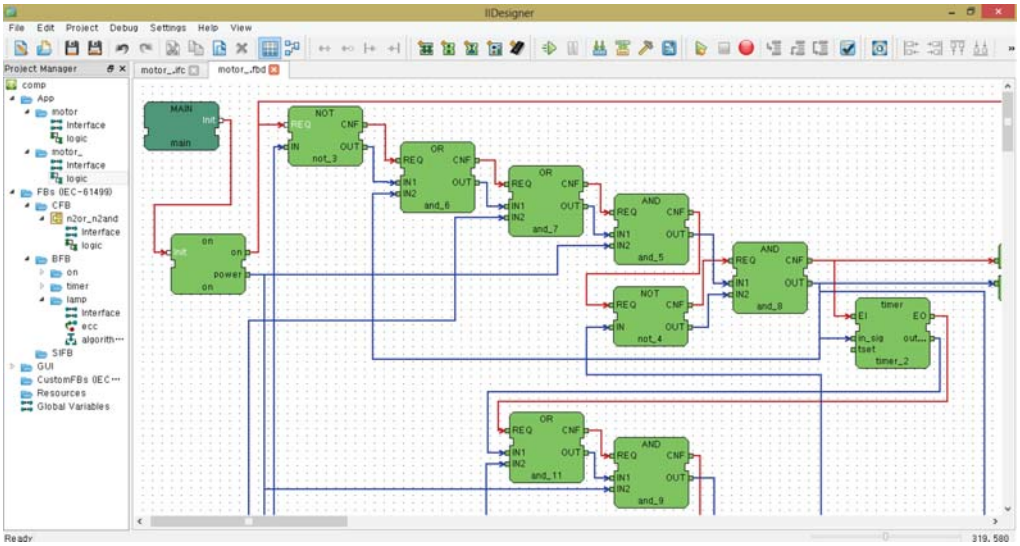

Figure 5. Motor operation example by the previous model using IIDesigner.

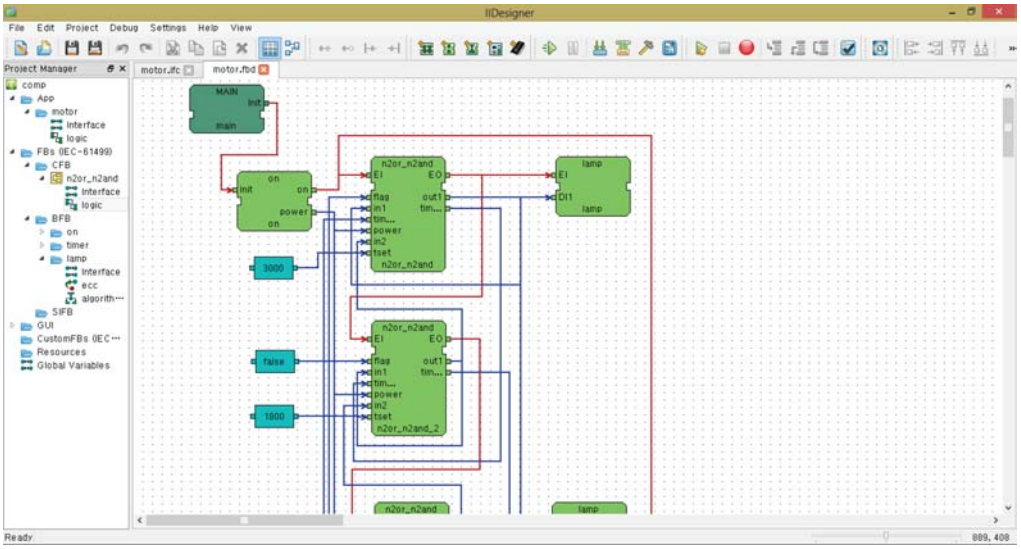

Figure 6. Motor operation example by the proposed model using IIDesigner.

Table 1 shows comparison results on structural representation compared with the previous model [19]. As shown in Table 1, the proposed model can correctly represent the applications with the hierarchical structure by presenting the formal definition of the composite function block.

Table 1. Comparison results on structural representation

Method	BFB	SIFB	CFB	Application	Subapplication
Previous model	○	○	×	○	×
Proposed model	○	○	○	○	○

(○: supported, ×: not supported)

Table 2 shows the functional comparison results compared with execution models, BSEM, NPMTR, and CBEM, for IEC 61499 applications in [20]. As shown in Table 2, the proposed model can correctly analyze the execution process for IEC 61499 applications with composite function blocks by defining newly the execution functions on the composite function blocks compared with previous models.

Table 2. Functional comparison results

Method	Instance queue	Event queue	Boolean variable	BFB	SIFB	CFB
BSEM	○	○	×	○	○	×
NPMTR	×	×	○	○	○	×
CBEM	×	○	×	○	○	×
Proposed model	○	○	×	○	○	○

Table 3 shows the size comparison results compared with previous models [19]. Columns 2-4, respectively, show the number of blocks and connections, and code size in previous models. Columns 5-7, respectively, show the number of blocks and connections, and code size in proposed models. As shown in Table 3, the proposed model can simply represent the structure of the application and reduce the size of code since a number of function blocks and connections is small compared with previous models.

Table 3. Size comparison

Application	Previous model			Proposed model		
	blocks	connections	Size (KB)	blocks	connections	Size (KB)
Motor operation	26	46	3	9	18	2
Fountain control	43	79	4	21	42	3

A comparison of execution time is shown in Table 4. As shown here, the execution time of the applications using the proposed model is almost similar to the previous model.

Table 4. Performance comparison (time in milliseconds)

Method	Motor operation	Fountain control
Previous model	54.0	206.4
Proposed model	55.0	213.2

Overall, the proposed model can be simply and hierarchically represented the distributed control applications and can reduce the size of execution program and can be effectively used in execution analysis of the applications composed of different kinds of function blocks including composite function block. A side-effect of using the proposed model is that a subapplication corresponding to the composite function block must be configured. The IIDesigner compiler generates subapplication corresponding to composite function block, which is standard practice. Designers can change this subapplication by editing the composite function block. The use of subapplication corresponding to composite function block can increase the runtime of software, but this is almost similar to the previous model. However, it can reduce the compiled code size to use the composite function block and can be simply and hierarchically represented the software structure.

5. Conclusion

Application of the distributed control system for factory automata composes of different kinds of function blocks. However, the increment of the fields of application system complicates the structure and behaviour of application extremely. Therefore, if it uses the proposed formal model for the application, it extremely facilitates to perform the complex control logic and can improve the efficiency of the software development. This paper has the advantage of as follows by presenting the formal definition of the composite function blocks. First, the proposed model can simply represent the structure of the overall software of the distributed control systems. Second, the proposed model can reduce the size of the execution program. Third, the proposed model can correctly analyze the execution process of the IEC 61499 applications with composite function blocks. The proposed model will be effectively used to analyze the runtime environment of the software tools for designing and developing of the complex application.

References

[1] A. Zoitl and V. Vyatkin, IEC 61499 architecture for distributed automation: The "glass half full" view, IEEE Industrial Electronics Magazine 3(4) (2009), 7-23.

DOI: https://doi.org/10.1109/MIE.2009.934789

[2] C. Gerber, H.-M. Hanisch and S. Ebbinghaus, From IEC 61131 to IEC 61499 for distributed systems: A case study, EURASIP Journal on Embedded Systems, Article 231630 (2008), 1-8.

[3] K. Thramboulidis, IEC 61499 vs. 61131: A comparison based on misperceptions, Journal of Software Engineering and Applications 6(8) (2013), 405-415.

DOI: https://doi.org/10.4236/jsea.2013.68050

[4] N. Kashyap, C. Yang, S. Sierla and P. G. Flikkema, Automated fault location and isolation in distribution grids with distributed control and unreliable communication, IEEE Transactions on Industrial Electronics 62(4) (2015), 2612-2619.

DOI: https://doi.org/10.1109/TIE.2014.2387093

[5] G. Zhabelova, V. Vyatkin and V. N. Dubinin, Toward industrially usable agent technology for smart grid automation, IEEE Transactions on Industrial Electronics 62(4) (2015), 2629-2641

DOI: https://doi.org/10.1109/TIE.2014.2371777

[6] P. Lindgren, J. Eriksson, M. Lindner, A. Lindner, D. Pereira and L. M. Pinho, End-to-end response time of IEC 61499 distributed applications over switched ethernet, IEEE Transaction on Industrial Informatics 13(1) (2017), 287-297.

DOI: https://doi.org/10.1109/TII.2016.2626463

[7] L. I. Pinto, C. D. Vasconcellos, R. S. U. Rosso and G. H. Negri, ICARU-FB: An IEC 61499 compliant multiplatform software infrastructure, IEEE Transactions on Industrial Informatics 12(3) (2016), 1074-1083.

DOI: https://doi.org/10.1109/TII.2016.2549862

[8] F. Andren, R. Brundlinger and T. Strasser, IEC 61850/61499 control of distributed energy resources: Concept, guidelines, and implementation, IEEE Transactions on Energy Conversion 29(4) (2014), 1008-1017.

DOI: https://doi.org/10.1109/TEC.2014.2352338

[9] T. Peng, X. Xu and L. Wang, A novel energy demand modelling approach for CNC machining based on function blocks, Journal of Manufacturing Systems 33(1) (2014), 196-218.

DOI: https://doi.org/10.1016/j.jmsy.2013.12.004

[10] G. Čengić, O. Ljungkrantz and K. Åkesson, Formal modeling of function block applications running in IEC 61499 execution runtime, in Proceedings of the 11th IEEE International Conference on Emerging Technologies and Factory Automation (2006), 1269-1276.

DOI: https://doi.org/10.1109/ETFA.2006.355187

[11] S. Panjaitan and G. Frey, Functional design for IEC 61499 distributed control systems using UML activity diagrams, Proceedings of the 2005 International Conference on Instrumentation, Communications and Information Technology ICICI 2005, Bandung, Indonesia (2005), 64-70.

[12] M. Fletcher and R. W. Brennan, Designing holonic manufacturing systems using the IEC 61499 (function block) architecture, IEICE Transactions on Information and Systems E84D(10) (2001), 1398-1401.

[13] C. Sunder, A. Zoitl, J. H. Christensen, M. Colla and T. Strasser, Execution models for the IEC 61499 elements composite function block and subapplication, 5th IEEE International Conference on Industrial Informatics (INDIN'07), Vienna, Austria (2007), 1169-1175.

DOI: https://doi.org/10.1109/INDIN.2007.4384941

[14] W. J. Kim and S. I. Cha, Research on the structural model of IEC 61499 applications with composite function blocks, Bulletin of the Academy of Science, the DPR Korea 375(3) (2017), 25-26.

[15] G. D. Shaw, P. S. Roop and Z. Salcic, A hierarchical and concurrent approach for IEC 61499 function blocks, 2009 IEEE Conference on Emerging Technologies & Factory Automation (ISSN: 978-1-4244-2728-4), 1 (2009), 1-8.

DOI: https://doi.org/10.1109/ETFA.2009.5347020

[16] R. Sinha, P. S. Roop, G. Shaw, Z. Salcic and M. M. Y. Kuo, Hierarchical and concurrent ECCs for IEC 61499 function blocks, IEEE Transactions on Industrial Informatics 12(1) (2016), 59-68.

DOI: https://doi.org/10.1109/TII.2015.2496262

[17] W. Dai, V. N. Dubinin and V. Vyatkin, Migration from PLC to IEC 61499 using semantic web technologies, IEEE Transactions on Systems, Man, and Cybernetics: Systems 44(3) (2014), 277-291.

DOI: https://doi.org/10.1109/TSMCC.2013.2264671

[18] V. Dubinin and V. Vyatkin, On definition of a formal model for IEC 61499 function blocks, EURASIP Journal Embedded Systems (2008), 1-10.

DOI: https://doi.org/10.1155/2008/426713

[19] G. Čengić and K. Åkesson, On formal analysis of IEC 61499 applications, Part A: Modeling, IEEE Transactions on Industrial Informatics 6(2) (2010), 136-144.

DOI: https://doi.org/10.1109/TII.2010.2040392

[20] G. Čengić and K. Åkesson, On formal analysis of IEC 61499 applications, Part B: Execution semantics, IEEE Transactions on Industrial Informatics 6(2) (2010), 145-154.

DOI: https://doi.org/10.1109/TII.2010.2040393

[21] J. Carlson and L. Lednicki, Timing Analysis for IEC 61499, Version 1.0 (2012), 1-19.

[22] D. L'Her, P. L. Parc and L. Marcé, Proving sequential function chart programs using timed automata, Theoretical Computer Science 267(1-2) (2001), 141-155.

DOI: https://doi.org/10.1016/S0304-3975(00)00301-7

[23] J.-R. Beauvais, E. Rutten, T. Gautier, R. Houdebine, P. L. Guernic and Y.-M. Tang, Modeling statecharts and activitycharts as SIGNAL equations, ACM Transaction on Software Engineering Methodology 10(4) (2001), 397-451.

DOI: https://doi.org/10.1145/384189.384191

■