Discrete-Event-Based Deterministic Execution Semantics With Timestamps for Industrial Cyber-Physical Systems

Wenbin Dai, *Senior Member, IEEE*, Cheng Pang, *Member, IEEE*, Valeriy Vyatkin, *Senior Member, IEEE*, James H. Christensen, and Xinping Guan, *Senior Member, IEEE*

***Abstract*—Cyber-physical systems (CPSs) are becoming common in the industrial automation domain. In industrial CPS, distributed programmable logic controllers collaborate to control manufacturing plants. Design and analysis of such systems require a system model that covers control, computation, and communication with physical plant dynamics. This paper focuses on execution semantics for industrial CPS with the aim of providing a deterministic and platform-independent execution environment. A discrete-event-based execution semantics augmented with timestamp mechanism is proposed for IEC 61499 to provide deterministic behavior and guarantee compliance with real-time constraints for industrial CPS. The timestamped discrete-event-based execution semantics is implemented in an IEC 61499 runtime with service-enabled features. A case study of building automation system is used to prove the proposed semantics.**

***Index Terms*—Building automation systems, deterministic execution semantics, IEC 61499 function blocks, industrial cyber-physical systems, real-time constraints, timestamped discrete-event systems.**

I. Introduction

CYBER-PHYSICAL systems (CPSs) are used in many applications of information and communication technologies (ICTs) and characterized by tight integration and counter influences between control, computation and communication phenomena [1]. Industrial automation is one primary application domain of CPS. In a nutshell, the CPS concept offers an integrated view of control, computation, and communication as well as physical processes dynamics for system design and analysis [2]. Since computation could reside within every component in CPS, it is not sufficient to understand individual properties of computational components. Moreover, the interaction between various physical components must be considered.

The design of CPS imposes challenges related to time and event-driven control, variable time delays, failures, reconfiguration, and distributed decision support systems [1]. Therefore, the design of such systems requires a holistic understanding of the interacting dynamics of computers, software, networks, and physical processes [2]. The CPS approach has shown its benefits in domains such as aerospace engineering, military applications, automotive, healthcare, and transportation. CPS also show impacts on the industrial automation domain from process control to manufacturing systems [3].

With current advancements in ICT such as increased computing power, the size of memory storage and network bandwidth, complexities of modern industrial automation systems are continuously increasing. Ethernet technology is widely adopted that has given a substantial boost to distributed industrial automation systems [4]. Control software requires an adequate model to cover the diversity of distributed components in industrial CPS. Many of these challenges could be solved by the IEC 61499 standard [5].

The IEC 61499 standard defines software components for distributed automation systems. The programming organization units are designed as event-triggered function blocks (FBs). Algorithms could be written in any programming languages such as IEC 61131-3 LD, ST, or even high-level programming languages including C++ and Java. Complexities are hidden from users by encapsulation of FB implementations and the use of predefined interfaces. FBs are categorized as basic FB [(BFB) state machines], service interface FB [(SIFB) platform dependent implementations], and composite FB (CFB)/subapplication (encapsulating function networks). This standard provides an abstraction model by encapsulating control logic into nested FBs and hides complexities from users through different levels of hierarchy. Furthermore, IEC 61499 models are directly executable, in contrast to other modeling languages, e.g., SysML [6] and AutomationML [7], which require additional steps before the executable code can be generated.

As far as the execution is concerned, two characteristics industrial CPS are important. First, execution semantics must

Manuscript received May 8, 2017; accepted July 24, 2017. This work was supported in part by the National Science Foundation, China, under Grant 61503247, in part by the Shanghai Pu Jiang Scholarship 2015 under Grant 15PJ1403400, and in part by the DIACPA Project of Swedish Research Council (Vetenskapprådet). This paper was recommended by Associate Editor R. Qiu. *(Corresponding author: Wenbin Dai.)*

W. Dai is with the Department of Automation, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: w.dai@ieee.org).

C. Pang is with Googol Technology (Shenzhen) Ltd., Shenzhen 518057, China, and also with Jiangmen Goobotics Research Institute, Jiangmen 529000, China (e-mail: cheng.pang.phd@ieee.org).

V. Vyatkin is with the Department of Computer Science, Electrical and Space Engineering, Luleå University of Technology, 97 187 Luleå, Sweden.

J. H. Christensen is with the Department of Research and Development, Holobloc Inc., Cleveland, OH 44121 USA.

X. Guan is with the Department of Automation, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: xpguan@sjtu.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMC.2017.2736339

be able to handle different *real-time constraints* imposed by control of physical phenomena. In legacy industrial automation systems, real-time requirements have been addressed by choosing hardware with necessary performance. Hence, identical control software can cause unexpected behaviors due to variations of execution times on multiple hardware platforms. As programs are scheduled in sequential order, real-time constraints cannot be guaranteed if the system deployment plan is modified. Second, execution semantics is expected to guarantee *determinism*. Determinism can be achieved if, with given initial states and sequences of input values, system output is always predictable given any elapsed time. Same output results must be reproduced with same initial states, input value sequences, and time elapsed. However, distributed control nodes often operate asynchronously in existing industrial automation systems. As a result, it is sometimes an extremely challenging task to repeat a site failure in a simulation environment. Overall, this paper aims to propose a deterministic execution semantics that is totally independent of deployment platforms.

The remaining of this paper has the following structure. In Section II, all existing execution semantics and computational models of IEC 61499 are investigated. In Section III, an execution semantics based on discrete-event systems (DESs) is proposed for industrial CPS. The proposed model is further extended with real-time capabilities by inserting timestamps in Section IV. In Section V, the semantic rules are formally described for the execution model. In Section VI, timing analysis techniques for industrial CPS are provided. Following that, an IEC 61499 FB execution environment based on the proposed model is implemented and details are given. A case study of building automation system is used to illustrate the proposed execution model in Section VIII. Also, issues found during implementation and testing regarding the proposed execution model are discussed in Section IX. Finally, this paper is concluded with future works are recommended.

II. Related Works

An industry-academic partnership *iCyPhy* has been established to promote research for CPS across industries [3]. The *iCyPhy* consortium focuses on design methodologies, heterogeneous modeling, and tool integration as the major concerns for industrial CPS design.

Eidson *et al.* [8] proposed a programming model, namely PTIDES, for generic design and coordination language for CPS. The PTIDES model offers a robust and hardware independent modeling language for distributed real-time system-level software development. PTIDES also provides deterministic execution semantics, in which variability in clock synchronization and network latencies can be eliminated from the physical plant model of any CPS. The PTIDES model extends from DESs and uses an actor-oriented approach [9]. An actor is a reusable software component that represents a separated time domain [10]. Actor components are triggered concurrently by time-stamped input events and process input events in chronological order. The execution semantics of the PTIDES model relies on a tagged signal model. This model provides deterministic temporal semantics by using actors and domains [11]. When identical sets of input events with timestamps are provided, an actor is used for each domain to generate an identical set of output events as well as timestamps. Interactions of control programs and physical processes are represented in the same system model without specifying hardware details. As real-time constraints are met at sensors, actuators, and network interfaces, modifications in hardware details or small variations in program execution time will not affect the system behaviors [12]. The PTIDES model proves the feasibility of integrating a time-triggered compliment along with DESs.

Mixed time-triggered and event-triggered systems are a popular topic in industrial related research. Concepts of event-triggered and time-triggered systems are compared by Albert [13]. One can conclude that there is no jitter for event-triggered systems if all distributed nodes are synchronized to a global time. Event-triggered systems normally provide rapid responses to asynchronous external requests. However, time-triggered systems normally lack flexibility and scalability since a small modification may result in a complete system redesign process. Besides, a hybrid event-triggered and time-triggered system are proposed to provide better flexibility and scalability as well as improve performance. A limited pre-emptive scheduling method is proposed by Van Den Heuvel *et al.* [14] based on mixing time-triggered and event-triggered tasks. A global table is used for managing and dispatching time-triggered and event-triggered tasks. Timing constraints of nonpreemptive time-triggered tasks are ensured by a newly designed synchronous protocol. Both event-triggered and time-triggered systems are evaluated by Scarlett and Brennan [15] from the perspective of applying object-oriented paradigm in industrial control systems. The conclusions show that time-triggered systems have better dependability but event-triggered systems have more flexibility and fast response.

Pang *et al.* [16] and Vyatkin *et al.* [17] proposed a unified architecture for a time-complemented event-driven computational model for distributed control systems. The proposed model combined two fundamental design paradigms in industrial automation systems for synchronizing decentralized activities. This combination is proved to offer the expressiveness of event-driven programming and the determinism of time-driven logic. The IEC 61499 standard is the foundation of this symbiosis that provides event-driven distributed control architecture. The IEEE 1588 precision time protocol (PTP) [18] is adopted for establishing the basis for highly accurate time synchronization. The proposed time-complemented event-triggered distributed control model improves the reusability and flexibility of automation software with reasonable real-time performance [19].

Programmable logic controllers (PLCs) are widely deployed in almost every industrial automation systems. The cyclical execution semantics is used by all IEC 61131-3 compliance PLCs. Once execution starts, PLCs will repeat the scan cycle indefinitely. In each PLC scan cycle, PLCs read values of inputs first, then execute all active tasks in a sequence and finally update values of outputs [20].

There exists various execution semantics in the IEC 61499 standard for distributed industrial control [21], including cyclical execution [22]–[24], buffered sequential

execution [25], nonpreempted multithreading execution [26], and synchronous execution semantics [27].

Similar to the IEC 61131-3 model, the cyclical execution semantics for IEC 61499 is introduced by Tata and Vyatkin [22] and Vyatkin and Chouinard [23]. In each scan cycle, all FB instances are prescheduled in a fixed order of an application. In execution of each FB, data variables are updated from its interface initially. When an input event is received, logic encapsulated in this FB is executed. Once execution of algorithms is completed, updated output events and data variables are pushed to its interface. As FB instances are executed in sequential order, only one FB will be activated at any moment of time in cyclical execution.

The buffered sequential execution model [25] uses first-in-first-out (FIFO) to queue events. An FIFO-queue is used to store input events for each FB instance. Events are consumed as FIFO by this FB instance until no more input event exists in the FIFO-queue. Multithreading is used in buffered sequential execution that each concurrent thread is responsible for a separated FB chain [26]. When an FB chain is activated by an event source, this thread cannot be pre-empted until execution is terminated. In this case, FB networks are divided into several concurrent subnetworks that are scheduled individually.

The buffered sequential execution model and multithreading execution semantics are yet to be proven deterministic. Events may end up with different orders in an event queue as execution time varies, especially when events are passed from distributed nodes. In contrast, a synchronous execution semantics is proposed by Yoong *et al.* [27] for IEC 61499 FBs. All FB instances in synchronous semantics are executed based on clock ticks. No input event queue is required for each FB instance. In each tick, all FB instances are executed once. However, if more than one event is received at an FB instance in one tick, only the last event will be processed. The synchronous execution is deterministic as proven by Yoong *et al.* [27]. However, system reliability issues may arise when simultaneous events occur on the same tick since only one event will be queued for the next tick. Besides, the synchronous semantics rests on the assumption (synchrony hypothesis) that a tick is instantaneous which may be challenging to guarantee in CPS. Its performance is not independent of properties of hardware, relying on the best effort in fulfilling the synchrony hypothesis.

There are also some relevant works that provide valuable insights regarding execution for IEC 61499. Strasser *et al.* [28] discussed several execution issues of CFBs and subapplications and propose two execution models for CFBs. The first option is to execute CFBs as BFBs where execution cannot be interrupted. The second option is to flatten CFB structure so only BFBs and SIFBs are executed. Dubinin and Vyatkin [29] proposed design patterns which ensure robustness of FB execution. A one-to-one mapping between design patterns and execution models are illustrated. Lapp and Hanisch [30] proposed a DES synthesis methodology for controllers on the shop-floor level. This approach takes advantage of event interconnections using event transition invariants for Petri net models.

Zhabelova *et al.* [31] introduced the framework of cyber-physical components for industrial automation that is based on the IEC 61499 architecture and applied in smart grid applications. The proposed architecture provides a framework for integration of control, communication, and plant modeling, but does not consider the execution semantics. This paper attempts to fill this gap. Drozdov *et al.* [32] proposed a formal model for describing distributed automation CPS with CP-Agnostic software. The proposed model provides a guideline of distributed software model in CPS.

To summarize, despite the extensive investigations in this section, a comprehensive and well-accepted computational model is needed for industrial CPS that considers time-triggered along with event-triggered mechanism.

III. Discrete-Event Execution Model for IEC 61499 Function Blocks

DESs have been widely adopted in various domains. A DES is a discrete-state event-driven system whose state evolution entirely relies on the occurrence of asynchronous discrete events with time intervals [33]. A DES combines the time-driven system with the event-driven system frequently. State transitions in a time-driven system are commonly synchronized with clock ticks. State transitions only occur when events are raised in event-driven systems. State changes may happen at variable time intervals as events are generated asynchronously.

The main characteristic of the IEC 61499 standard is event-triggered FBs. An FB will only be executed when an event input is triggered. Events are generated from event source FBs (SIFBs) in IEC 61499 applications. For example, input data is polled at a fixed interval from industrial fieldbuses by SIFBs. Therefore, the DES can be fit into IEC 61499 computational model perfectly.

IEC 61499 DES-based execution semantics could be defined as follows. First of all, there is an event queue for each IEC 61499 *resource* in an IEC 61499 *device*. Given an IEC 61499 *resource* instance i, the event queue EQres is defined as

$$\mathrm{EQres}^i = \cup_{n=0}^{n\max} \mathrm{EI}n^i \tag{1}$$

where $n$max is the event queue size and EIn is an event element in the event queue.

All events are queued in an IEC 61499 resource in the chronological order as they arrive rather than creating an event queue for each FB instance. A circular FIFO buffer is used with a write pointer *wptr* and a read pointer *rptr*. Assuming transition time of events and variables is negligible inside an IEC 61499 resource, events will be extracted according to their orders in the queue.

As shown in Fig. 1, the discrete-event computational model is divided into three functions. First, given a new event ei, the input function Finput for the event queue is defined as

$$\mathrm{Finput} : \mathrm{EQres} \times ei \rightarrow \mathrm{EQres}'.$$

Algorithm 1 for the input function is given as follows.

The input function is responsible for ensuring there are free slots available in the queue, then writes arriving event into the

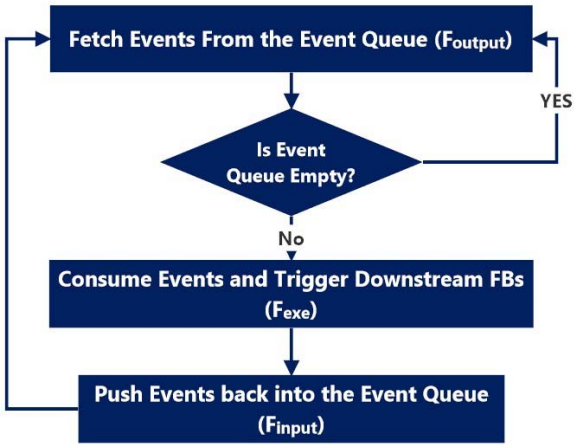


Fig. 1. Discrete-event computational model flow diagram.

Algorithm 1 Discrete Event Input Function

1: **if** *((wptr < rptr)* **and** *(rptr – wptr > 1))*
or *((wptr > rptr)* **and** *(wptr – rptr > 1))*
or *(wptr == rptr)* **then**
2: *EQres[wptr] = ei*
3: *wptr = (wptr + 1)* **mod** *nmax*
4: **end if**

Algorithm 2 Discrete Event Output Function

1: **if** *wptr <> rptr* **then**
2: *fetch EQres[rptr]* and *call the step function (see below)*
3: *rptr = (rptr + 1)* **mod** *nmax*
4: **end if**

next available slot and increments the write pointer. Similar to the input function, the output function Foutput for the event queue is defined as

$$\text{Foutput} : \text{EQres} \rightarrow \text{EQres}'.$$

Algorithm 2 for the output function is given as follows.

The output function is responsible for ensuring queue is not empty, then fetches the next event in the queue and calls the step function, and finally increments the read pointer. Finally, the execution of the FB network is described by the step function. The step function Fexe is stated as follows:

$$\text{Fexe} : \text{SSres} \rightarrow \text{SSres}'$$

1: SSres $\leftarrow$ Fh(EQres[*rptr*])
2: SSres$'$ $\leftarrow$ SSres
where *Fh* is the event handling function.

The step function takes a current set of events, processes the active event by triggering target FB instance, and invokes the input function to queue the output events emitted during the step.

IV. Introducing Timestamps Into Discrete-Event-Based IEC 61499 Execution Model

In the previous section, a discrete-event-based computational model is defined based on circular FIFO buffers. Continue from there, time-triggering concepts will be introduced into this model. As mentioned earlier, a time-stamped event is proposed by Lee [9] for CPS that each event consists of a data structure of deadline and micro-step. The proposed time-stamped event can be applied for industrial CPS as well. However, industrial automation systems have slightly different real-time constraints.

Events are generated by event sources FBs in IEC 61499 applications, for instance, SIFBs of analog and digital inputs via industrial fieldbuses or communication interfaces to other PLCs. Inputs and outputs on industrial fieldbuses are commonly scanned periodically. A PLC must provide responses to events from the physical plant before the next scan to ensure no overlap of execution occurs. When overlap happens, the PLC contains data from both scans. It will cause nondeterminism and loss of synchronization with the physical plant, which may lead to unexpected system behaviors. In CPS, tasks are executed within their deadlines to ensure on-time performance. For industrial CPS, algorithms are expected to be processed prior to the worst-case execution time (deadline) to ensure determinism.

To consider real-time constraints, a real-time clock must be introduced for providing accurate timestamps in DESs. The premise is made that all clocks are synchronized in the same IEC 61499 system configuration (for example, by applying IEEE 1588 PTP as described in [16]). Those synchronized clocks provide an unsigned number that represents the time elapsed since a system-defined reference time. An event *EIn* in (1) is defined as

$$EIn = (T_{\text{dl}}, T_{\text{last}}, P)$$

where T_{dl} is the deadline time when this event must be processed, T_{last} is the last time when this event was updated, and P is the priority of this event.

The deadline time T_{dl} represents the worst-case execution time when an event or its source must be processed in an IEC 61499 resource. The deadline time can only be updated by event source SIFBs. An event source SIFB could be any of event-related SIFBs such as E_RESTART, E_CYCLE, alternatively communication interfaces for industrial fieldbuses or message interpretation functions. Each event source FB will assign a deadline time according to its requirements. For instance, an I/O handling and E_CYCLE SIFB will set the deadline as its polling rate and E_RESTART will adjust the deadline time as "now" to force immediate execution. Once a deadline time stamp is set, this value will remain constant for all cascade events on the same event chain. As the event chain proposed in [26], an event chain starts from an event source FB and terminates when no further event output is generated from any FB directly connected from this event.

When an FB output event is emitted, the last time T_{last} will be refreshed with the current time from the system clock. If more than one event is emitted from the same FB simultaneously, those events will have identical last execution time value T_{last}. To distinguish the order of these events in the event queue, the priority P is then used for identifying simultaneous events emitted by the same FB, and $P \in \mathbb{N}$.

Algorithm 3 Input Function Ordered by Timestamps

```
1: if ((wptr < rptr) and (rptr – wptr > 1))
    or ((wptr > rptr) and (wptr – rptr > 1))
    or (wptr == rptr) then
2:   i = wptr - 1 //set search pointer starting position
3:   while i > rptr do
4:     if (EQres[i].Tdl < ei.Tdl)
   or ((EQres[i].Tdl == ei.Tdl) and (EQres[i].P < ei.P))
   or ((EQres[i].Tdl == ei.Tdl) and (EQres[i].P == ei.P) and
   (EQres[i].Tlast < ei.Tlast)) then
5:       EQres[wptr] = ei
6:       wptr = (wptr + 1) mod nmax
7:       end while
8:     else
9:       i = i – 1;
10:      if i < 0 then
11:        i = i + nmax
12:      end if
13:    end if
14:  end while
15: end if
```

As timestamps are embedded in events, the input function of the computational model proposed in the previous section must be modified. Instead of queuing events by chronological order when they appear in IEC 61499 resources, event outputs in time-stamped discrete-event execution semantics are ordered by following rules.

1) *Deadlines* are included in the event data structure.
2) If two or more event outputs with identical *deadlines* occur simultaneously, event outputs are further ordered by their *priorities*.
3) If two or more event outputs have identical *deadlines* as well as *priorities*, event outputs are sorted by last execution time in event data structure.
4) If two or more event outputs have identical timestamp data, event outputs are treated as a single event.

Algorithm 3 for input function Finput is given as follows.

Instead of using a simple FIFO buffer, events are inserted into the queue according to their chronological order. When a new event is generated, the input function will compare deadline, priority and last time execution of this event with all events in the queue to decide its position.

In the output function, events with the earliest combination of deadline and priority will be fetched from the queue. It is possible that more than one event output exists with the identical timestamp; in this case, the following rules apply.

1) If two or more event outputs are triggering a single FB, those events are merged into one request.
2) If two or more event outputs are triggering separate FBs, those FBs will be invoked in concurrent processes.

Algorithm 4 for the updated output function is given as follows.

V. Semantic Rules for Time-Stamped Discrete-Event-Based Execution Semantics

Time-stamped discrete-event-based execution semantics are defined as shown in the previous section. In this section, handling events within FB networks will be investigated.

Algorithm 4 Output Function With Timestamps

```
1: if wptr <> rptr then
2:   fetch EQres[rptr]
3:   if target FB is not invoked then
4:     call the step function
5:     rptr = (rptr + 1) mod nmax
6:   end if
7: end if
```

The step function Fexe of each IEC 61499 resource is shown in Fig. 2 below.

In the step function, the event-handling function with given input event *EI* will generate output event *EO* as

```
if target FB is BFB then
  for i = 0 to EO Count in the Active EC State
    EO = EI //No Communication Delay as Rule 5
    EO.P = i //Rule 3
    //Update Last Execution Time as Rule 2
    EO.Tlast = GetCurrentTime()
    EQres[wptr] = EO;
    wptr++;
  end for
else if target FB is SIFB then
  for i = 0 to EO Count in the Active Service Sequence
    EO = EI //Rule 5
    EO. Tdl = DT of this SIFB //Rule 1
    EO.P = i //Rule 4
    EO.Tlast = GetCurrentTime() //Rule 2
    EQres[wptr] = EO;
    wptr++;
  end for
else if target FB is SubApp or CFB then
  for i = 0 to with Event Connection Count with this
  EI
    EO = EI //Rule 5
    EO.P = i //Rule 8
    //Merge Event if duplicated as Rule 6
    if EO does not exist in EQres then
      EQres[wptr] = EO;
      wptr++;
    end if
  end for
end if
```

Several semantic rules are implemented in the event-handling function to provide deterministic system behaviors: given a set of initial states and sequence of input values, identical output values are always produced by systems. In this section, only semantic rules that is not stated in the IEC 61499 standard will be provided.

Rule 1: The deadline timestamp value of an event must be only set by event source SIFBs when it is emitted. The deadline timestamp value of an output event remains the same to its triggering input event. As defined earlier in this paper, the deadline timestamp can only be set by event source SIFBs such as E_RESTART, E_CYCLE, or communication interface SIFBs. For a BFB or a nonevent source SIFB, the deadline timestamp value of an emitted output event(s) will be identical to the triggering input event. For a CFB or subapplication,

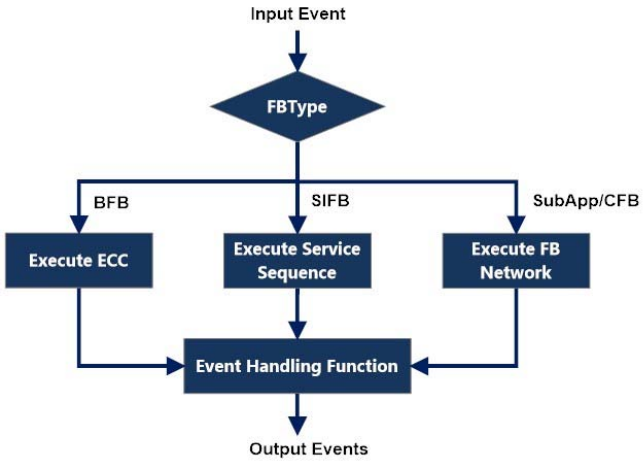


Fig. 2. Step function flow chart.

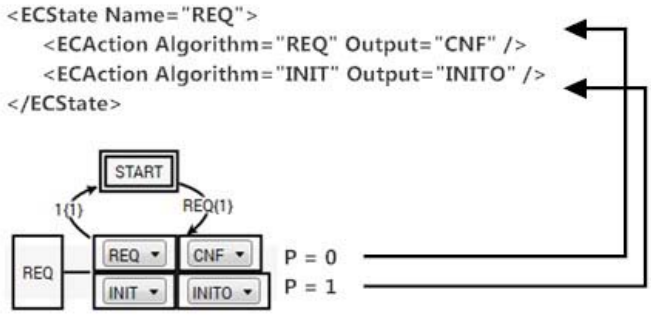


Fig. 3. Priorities in multiple EC state event outputs of BFB.

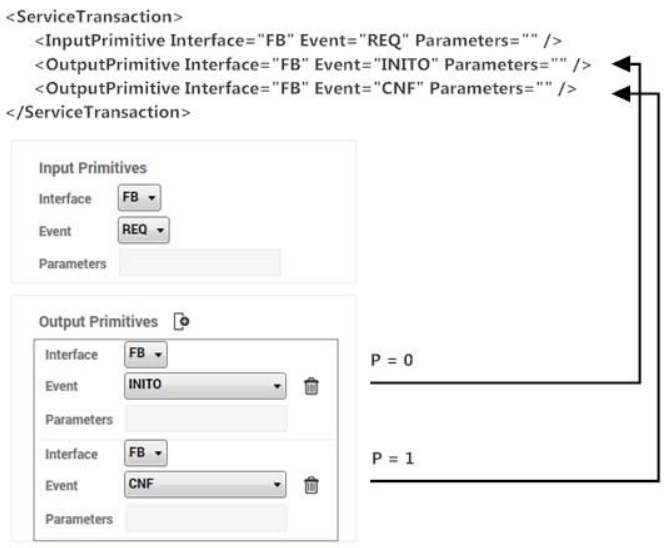


Fig. 4. Priorities in multiple service transaction event outputs of SIFB.

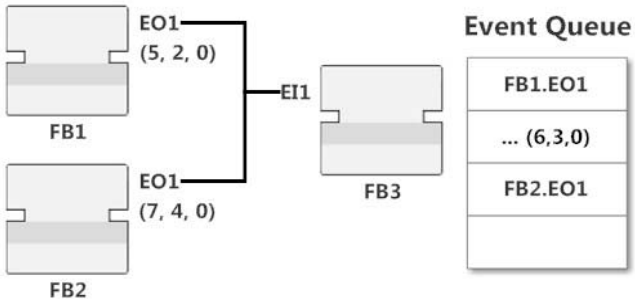


Fig. 5. Merging events in FB network.

the deadline timestamp value of an activated input event will be carried to output event(s) via internal events through the internal FB network (FBN).

Rule 2: For any FB type, the last execution timestamp value of an output event is updated with current system time when it is emitted from an FB. If there is more than one output event is emitted, then priority is set to indicate execution order. The last execution timestamp of an event shall only be updated at the output side of FB interface. The value of the last execution timestamp remains constant during propagation through event flows.

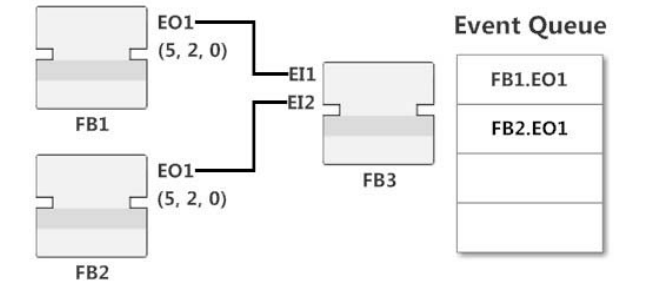
Rule 3: If there are two or more output events with identical last execution timestamps emitted from the same EC state in a BFB, priorities for these events are decided based on order as in their XML definitions. As shown in Fig. 3, an output event that is declared earlier in the XML definition will be assigned with higher priority. The first occurrence of the output event in the list will be given the highest priority (in this case, 0).

Rule 4: If there are two or more output events with identical last execution timestamps emitted from the same service sequence in an SIFB, priorities for these events are decided based on order as in their XML definitions. As illustrated in Fig. 4, an output event that is declared earlier in the output primitives will be given higher priority. The first occurrence of an output event will be assigned with the highest priority (0).

Rule 5: In a single resource, the event input of an event connection has identical timestamp and priority with the event output it is connected from. The propagation delay inside an IEC 61499 resource is assumed to be zero.

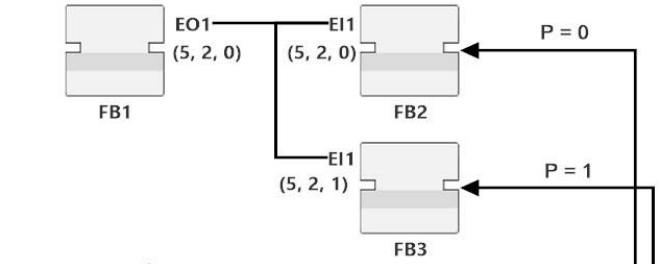
Rule 6: In an FBN, if two or more output events are merged into single input event, events will be consumed in their chronological order. If two or more output events have identical timestamps as well as priorities, those events will be merged into a single event. As shown in Fig. 5 below, a downstream FB may be triggered by several events from upstream FBs in short period of time. However, these events may not be consumed continuously of the downstream FB as there might be other events scheduled in between from other upstream FBs. If another event with identical timestamp and priority is found in the event queue, then new output received will be ignored.

Rule 7: If two or more output events are triggering two event inputs from the same FB in an FBN, events will be queued and consumed according to their chronological order. If two or more output events have identical timestamps as well as priorities, those events will be queued and consumed based on their occurrence in the XML definitions. As illustrated in Fig. 6 below, when two event outputs occur simultaneously, the downstream FB will be activated two or more times by various event inputs in their occurrence order in the XML file.




```
<EventConnections>
  <Connection Source="FB1.EO1" Destination="FB3.EI1" />
  <Connection Source="FB2.EO1" Destination="FB3.EI2" />
</EventConnections>
```

Fig. 6. Simultaneous events in FB network.




```
<EventConnections>
  <Connection Source="FB1.EO1" Destination="FB2.EI1" />
  <Connection Source="FB1.EO1" Destination="FB3.EI1" />
</EventConnections>
```

Fig. 7. Splitting events in FBN.

Rule 8: If two or more input events split from the same event output in an FBN, priorities of those input events are decided based on their orders in the XML definitions. As illustrated in Fig. 7 below, when an event output is connected to more than one FB event input, execution of downstream FBs will be scheduled in order of their occurrence in the XML file. Downstream FBs have identical timestamps but assigned with different priorities.

To summarize, the proposed time-stamped discrete-event-based execution semantics provides the determinism based on chronological orders presented by timestamps and priorities. A global event queue is introduced for every IEC 61499 resource to ensure that output events are placed in order. Although parallel execution is allowed in the IEC 61499 standard, an FB can be activated by only one input event at a time [28]. When an event output is linked to more than one FB instance, those FB instances would be invoked in parallel. When an event input is connected from two or more FB instances, this event input may be triggered multiple times depending on its timestamp and priority. To ensure determinism, the proposed semantic rules provide different levels of priorities for simultaneous events.

VI. Real-Time Constraints Analysis

In embedded systems research, real-time constraints analysis is always an important topic. The key index for embedded systems performance measurement is system reaction time to ensure "things are done at the right time." Regarding industrial CPS, this means the I/O polling cycle rate. As synchronous industrial fieldbuses are based on polling mechanism (except CAN bus which is based on message protocols), as long as execution of algorithms could be completed before the next I/O scan cycle, there is no performance gain.

If the event producing speed of the event source FB is faster than the consuming speed, the execution semantics will become nondeterministic. In this case, the event source FB emits another event before the previous event terminates on the same event chain. The new output event will be scheduled into the queue and executed prior to further downstream FBs. This could make unpredictable execution results of the FB network. To avoid nondeterministic execution, the execution environment must continuously monitor execution time to ensure that execution of algorithms could be completed within certain time constraints, for example, between two I/O scan cycles of PLCs.

Monitoring deadline is also considered in the proposed time-stamped discrete-event-based execution semantics. To ensure real-time constraints, at each event sink FB, the total execution time from event source FB is checked. An event sink FB is an FB that does not include any event outputs [26], such as SIFBs for updating output values back to fieldbuses. As described in the previous section, a deadline timestamp value is embedded in the event data structure. The margin of the execution time of this event chain $T\text{margin}$ is calculated by

$$T\text{margin} = ei.T\text{dl} - T\text{now}$$

where $T\text{now}$ is the current time obtained from clock and ei is input event to event sink FB.

As shown in Fig. 7, when execution of an event sink FB is accomplished, the margin time for this event chain could be calculated by subtracting the current timestamp from the deadline timestamp. If the margin time is less than zero, an alarm will be set for exceeding real-time constraints. The capacity of FB numbers placed in a system configuration can be identified instantly. To trace execution time of each FB, the last process time $T\text{last}$ in the timestamp can also be utilized as indicated in Fig. 7. Any extraordinarily long process time of an FB could be identified by subtracting last process times between the two connected FBs.

One important issue for IEC 61499 execution is to detect dangerous event loops. As described in Fig. 8 [34], dangerous event loops are loop-back events that could stop the system from responding. The proposed method for calculating margin time could also assist in the detection of nonresponsive systems. When an FB execution is terminated, margin time is calculated immediately as shown in Fig. 9. When the calculated margin time is significantly less than zero, and identical event sequence appears in the resource event queue, an alarm can be raised for dangerous event loops.

VII. Implementation

In the previous work [35], an IEC 61499 runtime, namely FB service runtime (FBSRT), is developed based

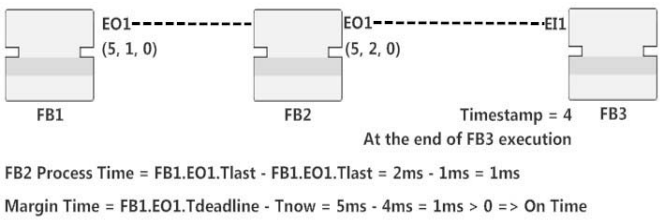


Fig. 8. Worst-case execution time measurement.

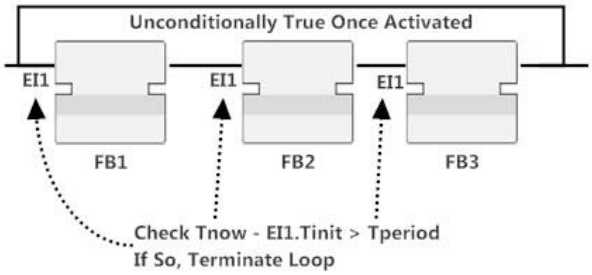


Fig. 9. Dangerous event-loop detection.

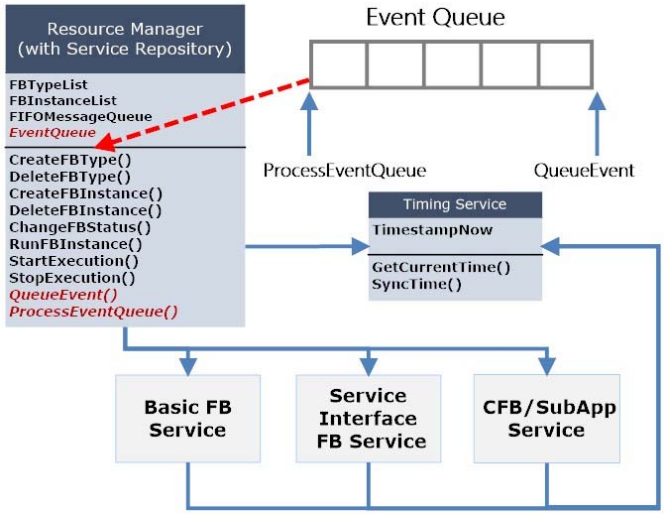


Fig. 10. Implementation for time-stamped discrete-event execution semantics in FBSRT.

on service-oriented architecture for bridging flexibility and interoperability in iCPS. In FBSRT, each FB instance is running as an independent software service that allows exchanging messages from other FBs. The FBSRT supports multiple hardware platforms and operating system combinations such as Intel/Windows, Intel/Linux, ARM/Linux, and Intel/QNX.

As shown in Fig. 10, the proposed time-stamped discrete-event executions semantics is implemented in the FBSRT. A resource event queue is introduced with a read and a write pointer in the resource manager. As mentioned earlier in this paper, two new services are added: 1) event input function *QueueEvent*()—for queuing events by chronological order and event output function *ProcessEventQueue*()—for processing next event in the FIFO queue. The event output function will continue processing events until no more event left in the

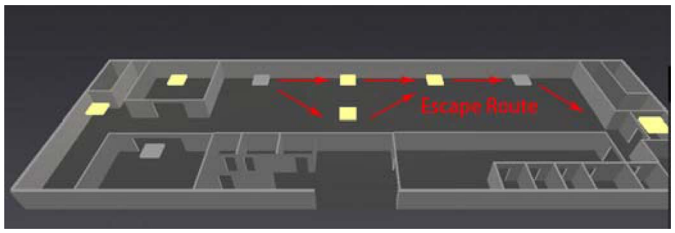


Fig. 11. Factory workshop lighting system floor plan.

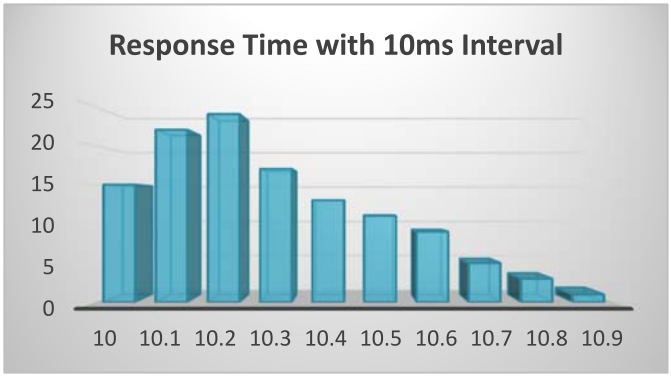


Fig. 12. Tests of response time with 10-ms interval.

queue. After executions are completed in BFB, SIFB, and CFB/SubApp, the generated events are pushed back to the main event queue by invoking the *QueueEvent*() function.

A new *timing* service is introduced into the FBSRT kernel. The *timing* service provides current system time for all other services in FBSRT by using the IEEE 1588 PTP [18]. The IEEE 1588 PTP is a protocol for synchronizing clock time over networked control systems based on a master–slave configuration, which has an accuracy at nanosecond level for all slave nodes. However, the timing of IEC 61499 resources could only be set at millisecond level due to time precision limitation of CPUs. For x86, x64, and ARM architectures, the minimum time scale is 1 ms. If a more precise time is required, routers with IEEE 1588-enabled could be used as the master clock to synchronize with slaves (for example, Cisco Stratix 5900 series [36]). When an update for any event is required by BFB, SIFB, and CFB/SubApp service, the timing service will return a current timestamp.

VIII. Case Study

The FBSRT with proposed execution model is tested with a lighting subsystem of the factory building automation system in a manufacturing facility. As shown in Fig. 11, each floor of the manufacturing plant consists of several light areas. Each light group is equipped with several LED ballasts, one environment illumination sensor. The DALI protocol [37] is used as fieldbus protocol for connecting ballasts and sensor.

As illustrated in Fig. 12, for each lighting group, an ARM-based PLC with Cortex-A8 AM335x 1-GHz CPU with 512-MB RAM, four analog inputs, and 12 analog outputs are used to provide ballast control. On each floor, an Intel x86-based PLC with a 1.4-GHz Intel Celeron CPU with 2-GB RAM onboard is deployed as supervisory control and

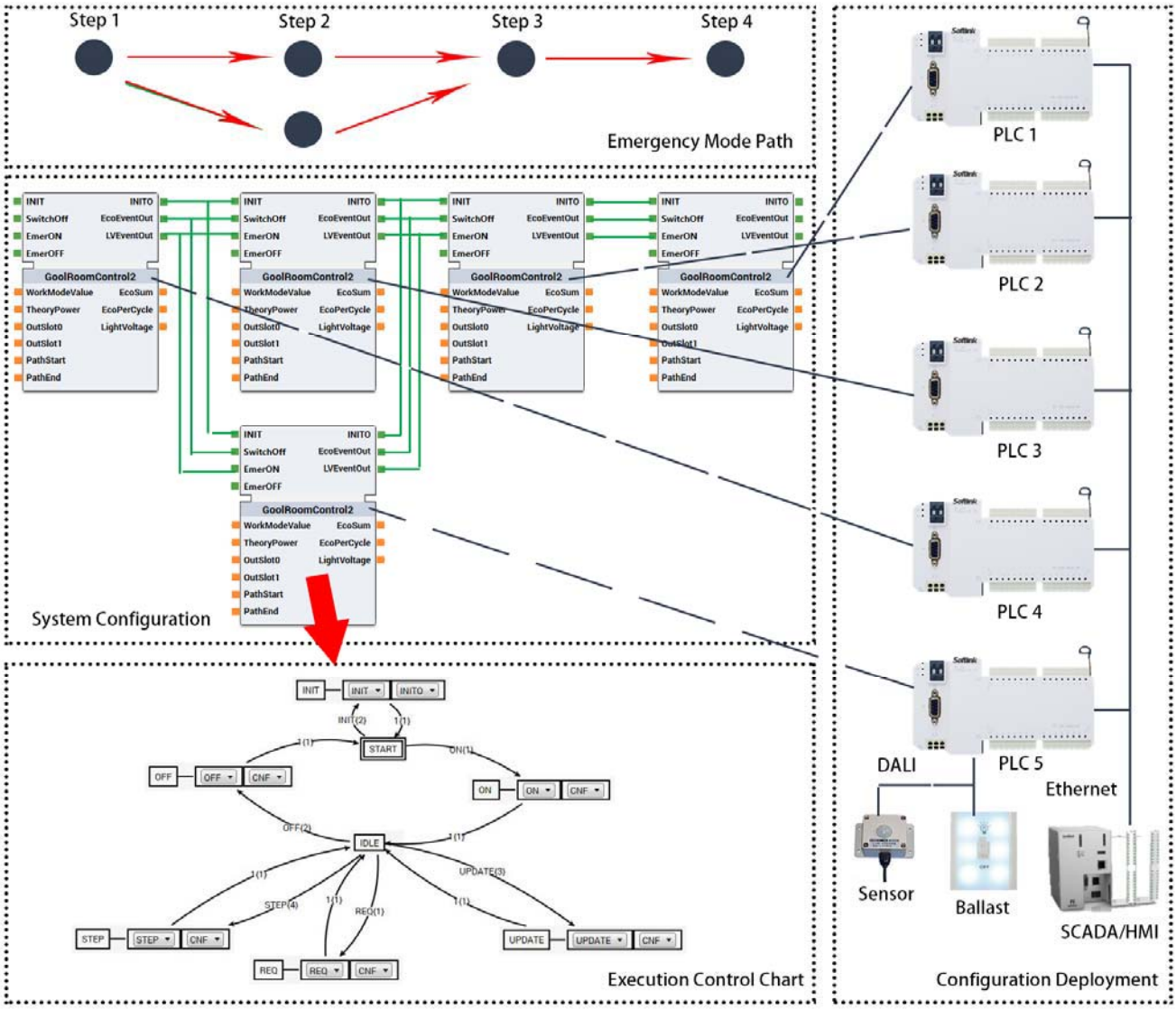


Fig. 13. Lighting system emergency mode IEC 61499 system design.

data acquisition (SCADA). Lighting control could be adjusted dynamically via a human–machine interface (HMI) which is also integrated with SCADA PLC. All PLCs are connected with each other using standard Ethernet cables.

Two tests were performed to verify the time-stamped discrete-event execution semantics with real-time constraints. For the deterministic test, the emergency mode is selected. In the emergency mode, all LED ballasts are indicating the direction to the exit of the building. These ballasts are light up one by one in a sequence. As indicated in Figs. 11 and 12, the escape path is divided into four steps and lights are flashing to guide workers escaping to the nearest emergency exit. The IEC 61499 system configuration for the floor lighting control is given in Fig. 13. For each lighting subgroup, a ballast control FB *RoomControl* is deployed on a separated PLC for providing features emergency escaping guide as well as other features supported by the DALI fieldbuses, for example, direct on/off and step up/down by time, etc.

To verify determinism, the ballasts are lit up in a fixed interval between steps which can be configured from SCADA/HMI. Two tests are performed with various interval values of 10 and 100 ms. As shown in Fig. 12 below, for 120 sample data of response time measured from PLCs with 10-ms interval between ballasts, the results prove that over 50% test data within 2% margin of time variation. All tests are completed within 10% margin of time variation.

With 100-ms interval, the results in Fig. 14 show that 35% test sample data with 1% margin and over 95% tests are within 5% margin of interval time.

For the platform independence test, all ARM-based light controllers are replaced with the Beckhoff CX2020 PLC that is equipped with a 1.4-GHz Intel Celeron CPU with 2-GB RAM onboard. The margin time then is recorded. From the results listed in Figs. 15 and 16, all margin times remain within 5% which means swapping controllers will not affect system behaviors as real-time requirements can be achieved by both controllers.

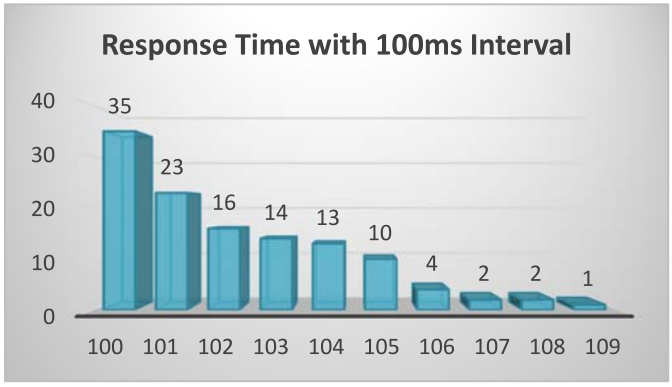


Fig. 14. Tests of response time with 100-ms interval.

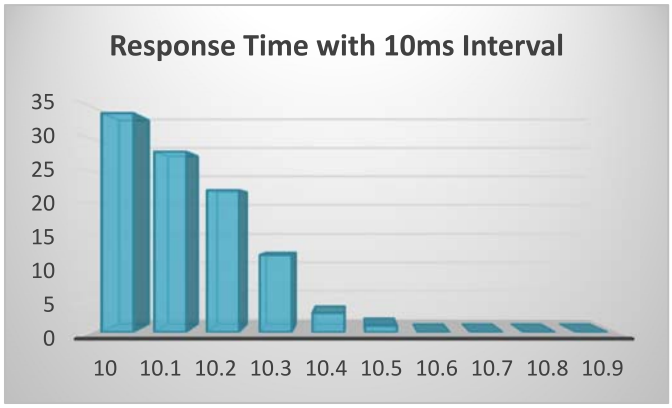


Fig. 15. CX2020 tests of response time with 10-ms interval.

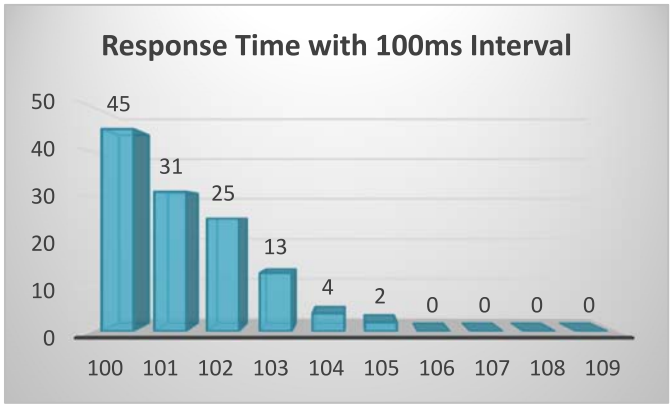


Fig. 16. CX2020 tests of response time with 100-ms interval.

IX. Discussion

Several findings obtained during the implementation process need to be discussed. First, as priorities of events also rely on orders in definition files (for example, BFB EC state output events and SIFB service sequences), different priorities must be set by IEC 61499 IDEs to avoid confusions from users when designing, developing, and testing systems. Users shall also be able to modify priorities in IEC 61499 IDEs to achieve deterministic system behavior.

Second, since events are extended as data structures rather than simple notifications, timestamps embedded in an event could be utilized in algorithms of BFBs and SIFBs. In existing IEC 61499 system design, timers are represented by separated SIFBs which must be placed outside BFBs. Once counting time is up, an extra event is used to notify BFBs. Alternatively, timer values can be passed into FBs as input variables, but this requires a huge amount of efforts. By using timestamped events, embedded deadline timestamps and last execution timestamps as variables could be directly used. The current system time is also available in algorithms offered by newly introduced *timing* service in the FBSRT. This will change design paradigms and bring benefits for time-constrained systems using IEC 61499.

Finally, the proposed time-stamped discrete-event execution semantics is based on a combination of sequential and parallel execution, which depends on how many FB instances an output event may trigger. Parallel execution could improve performance by dividing event chains into multiple concurrent threads. However, data consistency must be ensured at the design stage such that when a data variable is required by another parallel branch, it should not be time critical data (for example, a constant). As execution time varies for each parallel branch, there is no guarantee that this data variable will be updated before being used on an FB of another parallel branch.

X. Conclusion

Diversity is one typical characteristic of industrial CPS that various PLCs cooperate with each other via networks to provide intelligent control for complex industrial processes. A time-stamped discrete-event-based execution semantics is proposed for the IEC 61499 standard to meet different real-time constraints and provide deterministic execution behaviors in industrial CPS. The proposed execution semantics integrates timestamps with DESs and schedules execution of FB networks in chronological order. Deterministic execution is guaranteed by semantic rules of time-based event handling mechanism. The real-time constraints of industrial CPS are measured by monitoring the worst-case execution time continuously. Exceeds preset deadlines of the system reaction time may lead to unexpected behavior of industrial processes.

In future work, further experimentation will be carried out with the application of parallel execution semantics to the time-stamped discrete-event-based computational model. How to ensure deterministic execution semantics as well as how to improve performance by parallelizing the execution needs to be investigated. A pure event-driven update of inputs and outputs will be introduced. Furthermore, comparative performance analysis between the proposed execution semantics and other execution semantics needs to be taken. The invariance of CPS behavior regardless of properties of computational and communication platforms will be focused upon.

References

[1] R. Baheti and H. Gill, "Cyber-physical systems," in *Proc. IEEE Control Syst. Soc.*, 2011, pp. 161–166.

[2] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems, A Cyber-Physical Systems Approach*, 1st ed. Morrisville, NC, USA: LeeSeshia, 2011.

[3] P. Nuzzo, A. Sangiovanni-Vincentelli, and R. Murray, "Methodology and tools for next generation cyber-physical systems: The iCyPhy approach," in *Proc. 25th Annu. INCOSE Int. Symp. (IS)*, Seattle, WA, USA, 2015, pp. 235–249.

[4] V. Vyatkin, "Software engineering in industrial automation: State-of-the-art review," *IEEE Trans. Ind. Electron.*, vol. 9, no. 3, pp. 1234–1249, Aug. 2013.

[5] A. Zoitl and H. Prahofer, "Guidelines and patterns for building hierarchical automation solutions in the IEC 61499 modeling language," *IEEE Trans. Ind. Electron.*, vol. 9, no. 4, pp. 2387–2396, Nov. 2013.

[6] P. Pihlanko, S. Sierla, K. Thramboulidis, and M. Viitasalo, "An industrial evaluation of SysML: The case of a nuclear automation modernization project," in *Proc. 18th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Cagliari, Italy, 2013, pp. 1–8.

[7] R. Drath, "Let's talk AutomationML what is the effort of AutomationML programming?" in *Proc. 17th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Kraków, Poland, 2012, pp. 1–8.

[8] J. C. Eidson, E. A. Lee, S. Matic, S. A. Seshia, and Z. Jia, "Distributed real-time software for cyber–physical systems," *Proc. IEEE*, vol. 100, no. 1, pp. 45–59, Jan. 2012.

[9] E. A. Lee, "Modeling concurrent real-time processes using discrete events," *Ann. Softw. Eng.*, vol. 7, nos. 1–4, pp. 25–45, 1999.

[10] E. A. Lee, S. Neuendorffer, and M. J. Wirthlin, "Actor-oriented design of embedded hardware and software systems," *J. Circuits Syst. Comput.*, vol. 12, no. 3, pp. 231–260, 2003.

[11] Y. Zhao, J. Liu, and E. A. Lee, "A programming model for time-synchronized distributed real-time systems," in *Proc. 13th IEEE Real Time Embedded Technol. Appl. Symp. (RTAS)*, Bellevue, WA, USA, 2007, pp. 259–268.

[12] J. Zou, S. Matic, E. A. Lee, T. H. Feng, and P. Derler, "Execution strategies for PTIDES, a programming model for distributed embedded systems," in *Proc. 15th IEEE Real Time Embedded Technol. Appl. Symp. (RTAS)*, San Francisco, CA, USA, 2009, pp. 77–86.

[13] A. Albert, "Comparison of event-triggered and time-triggered concepts with regard to distributed control systems," in *Proc. Embedded World*, Nuremberg, Germany, 2004, pp. 235–252.

[14] M. M. H. P. Van Den Heuvel, R. J. Bril, X. Zhang, S. M. J. Abdullah, and D. Isovic, "Limited preemptive scheduling of mixed time-triggered and event-triggered tasks," in *Proc. 18th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Cagliari, Italy, 2013, pp. 1–9.

[15] J. J. Scarlett and R. W. Brennan, "Re-evaluating event-triggered and time-triggered systems," in *Proc. 11th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Prague, Czechia, 2006, pp. 655–661.

[16] C. Pang, J. Yan, and V. Vyatkin, "Time-complemented event-driven architecture for distributed automation systems," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 45, no. 8, pp. 1165–1177, Aug. 2015.

[17] V. Vyatkin, C. Pang, and S. Tripakis, "Towards cyber-physical agnosticism by enhancing IEC 61499 with PTIDES model of computations," in *Proc. Annu. Conf. IEEE Ind. Electron. Soc.*, Yokohama, Japan, 2015, pp. 001970–001975.

[18] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE Standard 1588-2008, 2008.

[19] C. Pang, W. Dai, and V. Vyatkin, "Towards IEC 61499 models of computation in Ptolemy II," in *Proc. Annu. Conf. IEEE Ind. Electron. Soc.*, Yokohama, Japan, 2015, pp. 1988–1993.

[20] *Function Blocks—Part 1: Architecture*, IEC Standard 61499-1, 2012.

[21] V. Vyatkin, "The IEC 61499 standard and its semantics," *IEEE Ind. Electron. Mag.*, vol. 3, no. 4, pp. 40–48, Dec. 2009.

[22] P. Tata and V. Vyatkin, "Proposing a novel IEC61499 runtime framework implementing the cyclic execution semantics," in *Proc. 7th IEEE Int. Conf. Ind. Informat. (INDIN)*, Cardiff, U.K., 2009, pp. 416–421.

[23] V. Vyatkin and J. Chouinard, "On comparisons of the ISaGRAF implementation of IEC 61499 with FBDK and other implementations," in *Proc. 6th IEEE Int. Conf. Ind. Informat. (INDIN)*, Daejeon, South Korea, 2008, pp. 289–294.

[24] F. Basile, P. Chiacchio, and D. Gerbasio, "On the implementation of industrial automation systems based on PLC," *IEEE Trans. Autom. Sci. Eng.*, vol. 10, no. 4, pp. 990–1003, Oct. 2013.

[25] G. Cengic and K. Akesson, "On formal analysis of IEC 61499 applications, part B: Execution semantics," *IEEE Trans. Ind. Informat.*, vol. 6, no. 2, pp. 145–154, May 2010.

[26] A. Zoitl, *Real-Time Execution for IEC 61499*, 2nd ed. Research Triangle Park, NC, USA: Int. Soc. Autom., 2009.

[27] L. H. Yoong, P. S. Roop, V. Vyatkin, and Z. Salcic, "A synchronous approach for IEC 61499 function block implementation," *IEEE Trans. Comput.*, vol. 58, no. 12, pp. 1599–1614, Dec. 2009.

[28] T. Strasser, A. Zoitl, J. H. Christensen, and C. Sünder, "Design and execution issues in IEC 61499 distributed automation and control systems," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 41, no. 1, pp. 41–51, Jan. 2011.

[29] V. N. Dubinin and V. Vyatkin, "Semantics-robust design patterns for IEC 61499," *IEEE Trans. Ind. Informat.*, vol. 8, no. 2, pp. 279–290, May 2012.

[30] H.-C. Lapp and H.-M. Hanisch, "A new DES control synthesis approach based on structural model properties," *IEEE Trans. Ind. Informat.*, vol. 9, no. 4, pp. 2340–2348, Nov. 2013.

[31] G. Zhabelova *et al.*, "Cyber-physical components for heterogeneous modelling, validation and implementation of smart grid intelligence," in *Proc. IEEE Conf. Ind. Informat.*, Porto Alegre, Brazil, 2014, pp. 411–417.

[32] D. Drozdov, S. Patil, and V. Vyatkin, "Formal modelling of distributed automation CPS with CP-agnostic software," in *Proc. Int. Workshop Service Orient. Holonic Multi Agent Manuf. (SOHOMA)*, Lisbon, Portugal, 2016, pp. 35–46.

[33] C. G. Cassandras and S. Lafortune, Eds., *Introduction to Discrete Event Systems*, 2nd ed. New York, NY, USA: Springer, 2008.

[34] W. Dai, V. N. Dubinin, and V. Vyatkin, "Automatically generated layered ontological models for semantic analysis of component-based control systems," *IEEE Trans. Ind. Informat.*, vol. 9, no. 4, pp. 2124–2136, Nov. 2013.

[35] W. Dai, V. Vyatkin, J. H. Christensen, and V. N. Dubinin, "Bridging service-oriented architecture and IEC 61499 for flexibility and interoperability," *IEEE Trans. Ind. Informat.*, vol. 11, no. 3, pp. 771–781, Jun. 2015.

[36] *Stratix 5900 Services Router*, Rockwell Autom., Milwaukee, WI, USA, Nov. 2013. [Online]. Available: http://ab.rockwellautomation.com/Networks-and-Communications/Stratix-5900-Services-Router

[37] *Digital Addressable Lighting Interface—Part 101: General Requirements—System*, IEC Standard 62386-101, 2014.

Wenbin Dai (GM'09–M'13–SM'16) received the B.Eng. (Hons.) degree in computer systems engineering from the University of Auckland, Auckland, New Zealand, in 2006 and the Ph.D. degree in electrical and electronic engineering from the Department of Electrical and Computer Engineering, University of Auckland, in 2012.

He is an Associate Professor with Shanghai Jiao Tong University, Shanghai, China. He was a Post-Doctoral Fellow with the Luleå University of Technology, Luleå, Sweden, from 2013 to 2014. He was also a Software Engineer from Glidepath Ltd., Auckland, New Zealand, from 2007 to 2013. His current research interests include IEC 61131-3 programmable logic controller, IEC 61499 function blocks, industrial cyber-physical systems, semantic Web technologies in industrial automation, and industrial software agents.

Cheng Pang (S'08–M'14) received the B.E. (Hons.) and M.E. (Hons.) degrees in computer systems engineering and the Ph.D. degree in electrical and electronic engineering from the University of Auckland, Auckland, New Zealand, in 2005, 2007, and 2013, respectively.

He is currently the Operation Deputy Manager of Googol Technology (Shenzhen) Ltd., Shenzhen, China, and the Associated Dean of Jiangmen Goobotics Research Institute, Beijing, China. His current research interests include model-driven engineering for industrial cyber-physical systems, building automation and control systems, and formal methods for industrial automation systems.

Valeriy Vyatkin (M'03–SM'04) received the Ph.D. degree from the State University of Radio Engineering, Taganrog, Russia, in 1992.

He is on joint appointment as a Chaired Professor of dependable computation and communication systems with the Luleå University of Technology, Luleå, Sweden, and a Professor of information and computer engineering in automation with Aalto University, Helsinki, Finland. He was a Visiting Scholar with Cambridge University, Cambridge, U.K., and had permanent academic appointments with the University of Auckland, Auckland, New Zealand, the Martin Luther University of Halle-Wittenberg, Halle, Germany, as well as in Japan and Russia. His current research interests include dependable distributed automation and industrial informatics, software engineering for industrial automation systems, and distributed architectures and multiagent systems applied in various industry sectors, including smart grid, material handling, building management systems, and reconfigurable manufacturing.

Dr. Vyatkin was a recipient of the Andrew P. Sage Award for the best IEEE TRANSACTIONS paper in 2012.

James H. Christensen received the Ph.D. degree in chemical engineering and computer science from the University of Wisconsin–Madison, Madison, WI, USA, in 1967.

He is currently with Holobloc Inc., Cleveland, OH, USA. He is an internationally recognized expert in the standardization and application of advanced software technologies to the automation and control of manufacturing processes.

Dr. Christensen was a recipient of the Rockwell International Engineer of the Year Award and the Lynde Bradley Innovation Award in 1991 for his achievements in pioneering applications of object-oriented programming in Smalltalk, and the IEC 1906 Award and Process Automation Hall of Fame Membership in 2007 for recognition of his accomplishments in the international standardization of programming languages and architectures for industrial automation.

Xinping Guan (M'02–SM'04) received the B.S. degree in mathematics from Harbin Normal University, Harbin, China, in 1986, and the M.S. degree in applied mathematics and the Ph.D. degree in electrical engineering from the Harbin Institute of Technology, Harbin, in 1991 and 1999, respectively.

He is currently a Professor with the Department of Automation, Shanghai Jiao Tong University, Shanghai, China. He is a Cheung Kong Scholars Programme Special Appointment Professor. He has (co)-authored over 200 papers in mathematical and technical journals, and conferences. His current research interests include functional differential and difference equations, robust control and intelligent control for time-delay systems, chaos control and synchronization, and congestion control of networks.