Time-Stamped Event based Execution Semantics for Industrial Cyber-Physical Systems

Wenbin Dai,
Shanghai Jiao Tong University, China,
IEEE Member, w.dai@ieee.org

Cheng Pang
Luleå University of Technology, Sweden,
IEEE Member, cheng.pang.phd@ieee.org

Valeriy Vyatkin,
Luleå University of Technology, Sweden and Aalto University, Finland, *Senior IEEE Member*,
vyatkin@ieee.org

James H. Christensen
Holobloc Inc, US
james.h.christensen@gmail.com

***Abstract* – Cyber-physical systems show their impacts in many areas including industrial automation. Design and analysis of cyber-physical automation systems requires an integral model that takes into account tight coordination of control, communication and physical plants dynamics, thus creating a new research domain, namely industrial cyber-physical systems. This paper proposes a new execution semantics for the IEC 61499 standard that is based on the concepts of discrete-event systems augmented with timestamping of events to guarantee real-time constraints for industrial cyber-physical systems. The proposed execution semantics is implemented in an IEC 61499 execution environment and a case study on building automation systems is used to verify the results.**

***Index Terms* —** *Industrial Cyber-Physical Systems; Discrete Event Systems; Timestamp; Execution Semantics; Deterministic; IEC 61499 Function Blocks; Real-time Constraints; Building Automation Systems.*

I. INTRODUCTION

Cyber-physical systems (CPS) is a popular research topic both in academia and industry motivated by tight integration between computation, control, communication and social interaction that is observed in many modern applications of information and communication technologies (ICT). Industrial automation is certainly one of such applications. In a nutshell, the CPS approach to systems design and analysis stipulates integrated view on computation, control, communication and physical processes dynamics. Since computing can reside within every physical component [1], it is not sufficient to understand properties of physical and computational components individually, it is important to understand their interaction [2].

Industrial automation systems experience high influence of advancements in ICT. With boosted memory size, computational performance, and network bandwidth, industrial automation systems are capable of handling complexities, which could not even be imagined a decade ago. More importantly, wide adoption of the Ethernet technology gave a strong boost to distribution of industrial automation logic [3]. Distributed control software asks for an adequate model that could cover the diversity of industrial CPS. The IEC 61499 standard has been developed to address many related challenges [4]. The IEC 61499 standard increases the modelling abstraction level by encapsulating control logic into function blocks and providing support for nested function blocks. Furthermore, IEC 61499 models are direct executable where extra steps must be taken for other modelling languages (e.g. SysML [5] and AutomationML [6]).

From an execution perspective, two characteristics are indispensable in industrial CPS. Firstly, execution semantics must address the *real-time constraints* of industrial CPS. In existing industrial automation systems, real-time performance largely relies on hardware capabilities. Hence, in such systems, identical programs could result in various system behaviours as execution times vary from platform to platform. Secondly, execution semantics must be *deterministic*. Deterministic execution means that with given initial states and sequences of input values, system status shall be predictable at a specified elapsed time and system operations could be repeated identically for identical initial states and input value sequences. Overall, the aim for this paper is to introduce an execution semantics that is deterministic and totally independent from deployment platforms.

The rest of the paper is organized as follows: In Section II, existing IEC 61499 execution semantics and models of computation investigated in CPS are revised. An execution semantics for industrial CPS based on discrete-event systems is introduced in Section III. In Section IV, the model of computation introduced is extended with real-time capabilities. Following that, the semantic rules for the computational model are defined in Section V. In Section VI, the implementation of the computational model for IEC 61499 function blocks is illustrated and experiences with this computational model are summarized in Section VII. Finally, conclusions and topics for future investigation are presented.

II. RELATED WORKS

In CPS research, Eidson et al. [7] proposed the PTIDES programming model, which is designed as a coordination language for CPS. PTIDES provides robust distributed real-time software models, which are independent of hardware deployment. The PTIDES semantics is based on a tagged-signal model, which provides deterministic temporal

semantics [8]. When presented with identical sets of input events with timestamps, an actor is required to deliver an identical set of output events with the same timestamps every time.

There also exist other works in industrial systems research regarding time-triggered systems and event-triggered systems. Albert [9] compares event-triggered and time-triggered concepts and concludes that if all distributed nodes are synchronized to a global time then there is no jitter for event-trigger systems. Event-triggered systems provide fast reactions to asynchronous external events. On the other hand, time-triggered systems lack of flexibility and scalability since a small modification requires a complete redesign of the system. A hybrid event-triggered and time-triggered system is suggested to provide better performance with improved flexibility and scalability. Van Den Heuvel et al. [10] proposes a limited pre-emptive scheduling method by mixing time-triggered and event-triggered tasks. A table is created for dispatching both time-triggered and event-triggered tasks. A synchronous protocol is designed for non-preemptive time-triggered tasks to ensure that timing constraints are satisfied. Scarlett et al. [11] evaluates event-triggered and time-trigger systems based on applying an object-oriented paradigm in industrial control systems. As a conclusion, time-triggered systems are more dependable and event-triggered systems are more responsive and flexible.

Pang et al. [12] propose a unified architecture for time-driven and event-driven control models, which are the two fundamental design paradigms applied in distributed control systems for synchronizing decentralized activities. The proposed time-complemented event-driven distributed control model aims at improving the modularity and flexibility of automation software with satisfactory control performance.

The majority of industrial automation systems are equipped with programmable logic controllers (PLC). There exists two international standard for PLCs: IEC 61131-3 [13] and IEC 61499 [14]. Execution semantics is clearly defined in the IEC 61131-3 standard, namely cyclical execution, in which PLCs repeat the scan cycle indefinitely. For each PLC scan cycle, PLCs read input values at the beginning of each scan, execute assigned tasks sequentially and update output values by the end of each scan. On the other hand, as execution semantics is not limited in the IEC 61499 standard, several execution semantics exist.

There is an IEC 61499 version of cyclical execution semantics introduced by [15] and [16]. On each scan, each function block is scheduled in a fixed order inside a function block network. When this function block is activated, input event and data variables are updated from its interface, encapsulated logic is executed, and finally all output event and data variables are written back to its interface.

The buffered sequential execution model [17] is based on FIFO-queues. For each function block instance, a FIFO-queue is used to store input events and events will be consumed by the function block instance until no input event is left in the queue. Multithreading execution consists of several concurrent threads in which each thread consists of a function block chain [18]. When a function block chain is triggered by an input event, this thread execution cannot be pre-empted once started. In this case, function block networks could be scheduled into multiple concurrent sub-networks.

Li et al. [19] propose a synchronous execution semantics for IEC 61499 function blocks in which all function block instances execute once when a clock tick is generated. There is no input event queue for each function block instance. If there is more than one event received at a function block instance during a tick, only one event will be stored for the next tick.

To conclude, despite the extensive investigations described above, there is not yet a comprehensive and well-accepted model of computation for industrial CPS that integrates a time-triggered mechanism with event-triggered systems.

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

Discrete event systems (DES) are widely adopted in various domains. A DES is a discrete-state event-driven system whose state evolution depends entirely on the occurrence of asynchronous discrete events over time [20]. A DES is frequently a combination of time-driven system and event-driven system. A time-driven system is triggered by clock ticks and state transitions are synchronized with the clock. In event-driven systems, state transitions only happen when events occur. Events are generated asynchronously which means state changes may happen at various time intervals.

The key characteristic of the IEC 61499 standard is event-triggered function blocks. A function block will only be activated when an event input from its interface is triggered. On the other hand, time-triggered events can be generated in various ways via SIFBs in IEC 61499 applications. For example, SIFBs can poll data at a fixed interval from industrial fieldbuses. Therefore, the DES concept could be perfectly fit into an IEC 61499 computational model.

Instead of queuing events at each function block instance, all events are queued in chronological order at resource level. The event queue is based on a circular FIFO buffer with a write pointer and a read pointer. Assuming transition time of event and data connections is negligible during execution in an IEC 61499 resource, events will be processed following their order in the queue. As queue data are specific for each resource, sets of these values must be stored which a process can take during the execution.

When events are writing into the queue, execution process checks whether any free slot is available in the queue, writes the event into the next available slot and increments the write pointer. When events are consumed from the queue, execution process checks whether the queue is empty, fetches the next event in the queue and calls the step function, and finally increments the read pointer. Finally, execution process takes current event queue, processes the active event, and updates current state of execution.

IV. Introducing Time into Discrete-Event IEC 61499 Function Block Network Execution Model

In the previous section, an event-triggered model of computation is defined based on discrete-event systems theory. In this section, this model will be augmented with time-triggering concepts. As discussed previously, a time-stamped event is introduced by Lee et al. [21] which contains a data structure of current time and priority for CPS. The proposed time-stamped event is also applicable for industrial CPS. However, real-time constraints for industrial automation systems must be considered.

In IEC 61499 applications, events are generated from external sources via SIFBs, for instance, analogue and digital inputs via industrial fieldbuses or messages from other PLCs. Inputs and outputs on industrial fieldbuses are commonly scanned periodically. PLCs must complete execution and update outputs between two scans to ensure no task overlap occurs. When task overlap happens, PLCs will be out of sync with fieldbuses, which may lead to unexpected system behaviours. In embedded systems, best-case reaction time is usually the key index for performance measurement. For industrial automation, worst-case execution time is constantly monitored in order to avoid non-deterministic execution.

In order to take real-time constraints into considerations, a real-time clock must be introduced for providing timestamps in discrete-event based execution semantics. The premise is made that all clocks in the same IEC 61499 system configuration are synchronized. Synchronized clocks provide a natural number that represents the time elapsed since a system-defined reference time. An event *EIn* defined in (1) is given a definition as:

$$EIn = (T_{init}, T_{last}, P)$$

where T_{init} is the original time when this event was created, T_{last} is the last time when this event was handled, and P is the priority of this event.

The initial time T_{init} refers to the first time when an event or its source appears in an IEC 61499 resource. The value of the initial time can only be set by event source SIFBs. An event source SIFB could be event-related SIFBs such as *E_RESTART*, *E_CYCLE,* or communication SIFBs for accessing fieldbuses and exchanging external messages. Once an initial time stamp is assigned, this value will be passed to cascade events on the same event chain. Similar to the event chain concept proposed in [18], an event chain starts from an event source function block and terminates when no further event output is generated from any function block connected by this event directly or through intermediate function blocks.

Whenever a function block output event is emitted, the last time stamp T_{last} will be updated with the current clock time. It is possible that more than one event is emitted from the same function block simultaneously, which means those events have identical values of last execution time T_{last}. In order to distinguish these events in the event queue, the priority P is introduced for identifying simultaneous events emitted by the same function block, and $P \in \mathbb{N}$.

Instead of following a simple first-in and first-out principal, events are listed in the queue by chronological order. When a new event is emitted, the input function will compare the last time execution timestamp of this event with all events in the queue in reverse order. This new event will be inserted into the queue at the position where the last execution timestamp of the previous event in the queue is earlier. If the last execution timestamps of two events are identical, the input function will terminate searching when the previous event has a higher priority. As events are already ordered according to the chronological order, the first event in the queue will always be the event with longest waiting time. There is no change required for the output function.

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

Time-stamped discrete-event based execution semantics are formally defined in previous sections. However, how to handle events is yet to be investigated. The step function, particularly event-handling function, will be described in this section. The event-handling function consists of several semantic rules. Those semantic rules aim to provide deterministic system behaviour: given the same initial state and sequence of input values, the system always produces identical output values. In this section, only semantics not covered in the IEC 61499 standard will be defined.

Firstly, initial timestamp value of an event is set only when it is emitted by event source SIFBs. As described in the previous section, the initial timestamp can only be set by event source SIFBs such as E_RESTART, E_CYCLE, or fieldbuses interface FBs.

Secondly, for any FB type, the initial timestamp value of an output event will be identical to that of its triggering input event. For a BFB or an SIFB, the initial timestamp value of an activated input event will be copied to the output event(s) emitted. For a CFB, the initial timestamp value of an activated input event will be carried by internal events through the internal FB network (FBN) and duplicated at output event(s).

Next, for any FB type, the last execution timestamp value of an event is updated with current clock time and priority is set when it is emitted from a function block. Last execution timestamp of an event will be only updated at function block right interface (output side). The last execution timestamp retains its values during propagation via event connections.

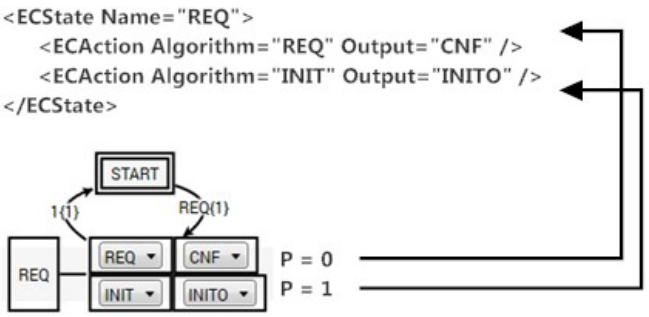

Fig. 1: BFB Simultaneous EC State Event Output

In a BFB, if there are two or more output events with identical last execution timestamps emitted from the same EC state, priorities for these events are set according to their orders in the XML file. As illustrated in Fig. 1, an output event that appears earlier in the XML file will be set with

higher priority. For instance, the priority of the first listed output event will be set to the highest priority, zero.

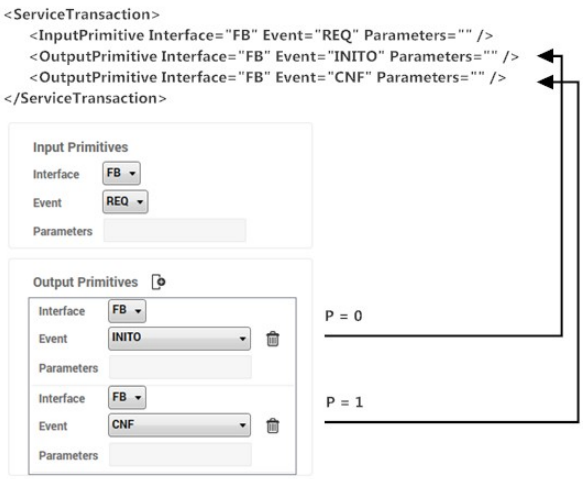

Fig. 2: SIFB Simultaneous Service Transaction Event Output

In an SIFB, if there are two or more output events with identical last execution timestamps emitted from the same service sequence, priorities for these events are set according to their orders in the XML file. As shown in the Fig. 2, an event that appears earlier in the output primitives will be set with higher priority. Again, priority of the first output event will be set to the highest priority, zero.

In an FBN, if there are two or more output events merged into one event input, events will be queued and processed by chronological order. The downstream function block may be invoked several times by upstream function blocks during one fieldbus scan cycle. However, the downstream function block is not necessarily activated continuously as there might be other events scheduled between events from upstream function blocks.

In an FBN, if there are two or more input events split from same event output, input event from event connection which appears earlier in the XML representation is set with higher priority. As shown in the Fig. 3 below, when an event output is connected to more than one FB event input, execution of downstream FBs will be scheduled in order of event connections placed in the XML file.

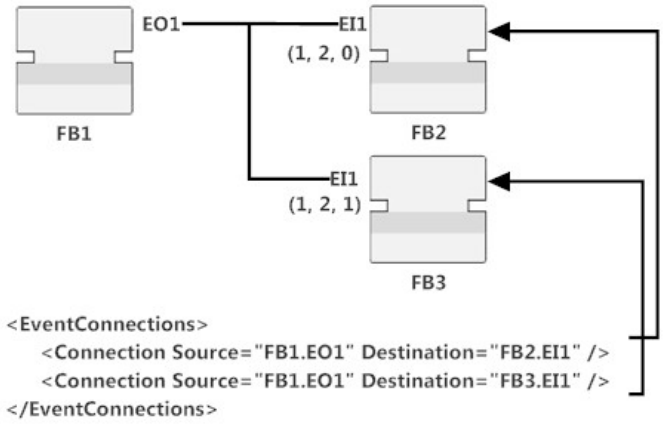

Fig. 3: FBN Event Split.

Overall, the determinism of the proposed time-stamped discrete-event based execution semantics is built based on sequential execution orders that are distinguished by timestamps and priorities. The one-dimensional array event queue introduced at the IEC 61499 resource level ensures that there is only one event activated simultaneously. Parallel execution is not limited in the IEC 61499 standard, as any function block shall only be activated by single input event simultaneously [22]. Although parallel execution would provide better performance, from the determinism perspective, the sequential execution is selected. The proposed semantic rules provide guidelines for assigning different priorities for simultaneous events ordered by time.

VI. Implementation

As described in [23], an IEC 61499 runtime Function Block Service Runtime (FBSRT) based on service-oriented architecture (SOA) is developed for bridging flexibility and interoperability. In FBSRT, each function block is running as an individual software service that communicates with other function blocks via messaging only. The proposed time-stamped discrete-event executions semantics is implemented in the FBSRT as illustrated in Fig. 4.

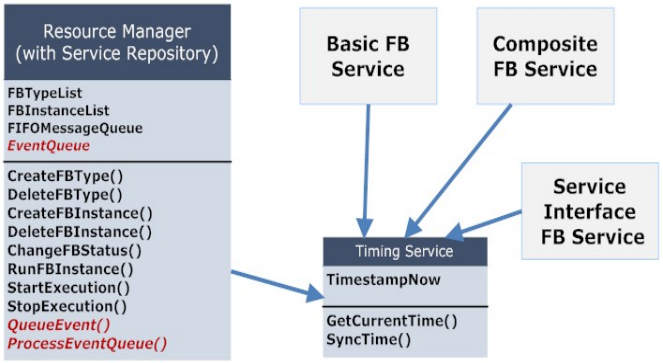

Fig. 4: FBSRT Implementation for Time-stamped Discrete-Event Execution Semantics.

In the resource manager, a new event queue is introduced as well as read and write pointers for the queue. As introduced in previous sections, two new functions are implemented: event input function *QueueEvent()* for queuing events by chronological order and event output function *ProcessEventQueue()* for process next event in the queue. The event output function will be invoked continuously until there is no more events left in the queue.

There is a new service inserted in the FBSRT known as timing service. The timing service offers current time for all other services in FBSRT by implementing the IEEE 1588 precision time protocol (PTP) [24]. The IEEE 1588 PTP is a precision time synchronization protocol for networked control systems based on a master-slave configuration. The IEEE 1588 PTP is capable of synchronizing all slave clocks in nanosecond level. However, due to time precision limitation on PLCs (on most x86, x64, and ARM architectures, the minimum time scale is one millisecond), time of IEC 61499 resources could be synchronized at millisecond level. If more precise time is required, IEEE 1588-enabled hardware could

be used for providing more accurate master clock time.

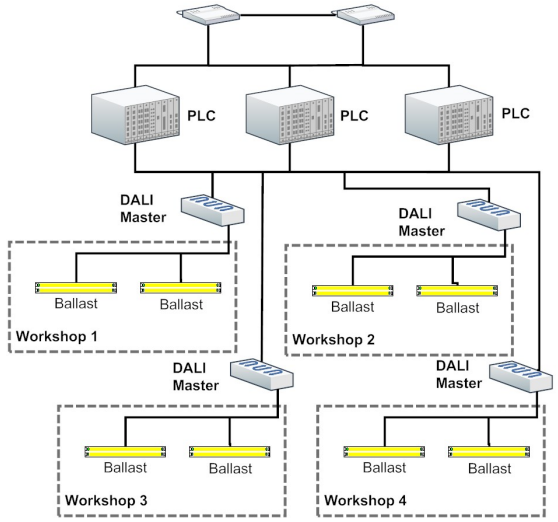

Fig. 5: Lighting Subsystem: Network Diagram.

The implementation is verified with a building automation system, in particular lighting subsystem of a manufacturing plant. As shown in Fig. 5, four workshops are located on this floor and each workshop has four ballasts. Ballasts in each room could be adjusted individually or by groups (2 or 4 in this case). The DALI protocol [25] is used for connecting ballasts with PLCs. For each workshop, a separated PLC is used and PLCs are inter-connected by standard Ethernet connections.

The IEC 61499 system configuration for this manufacturing plant is given in Fig. 6 below. For each workshop, a ballast control function block *FB_Ballast* is deployed for providing features supported by the DALI fieldbuses. For example, direct on/off and step up/down by time etc. A control panel is placed in each workshop that is represented by a SIFB *FB_CP*. Two SIFBs *FB_DALIIN* and *FB_DALIOUT* are used for accessing the DALI fieldbuses. A BFB *FB_Scenario* is designed to coordinate all workshops to perform a particular lighting scenario.

VII. Discussions

During the implementation processes, there are several findings need to be shared. First of all, as the proposed execution semantics affected by orders in XML file (for instance, orders of BFB EC state output events and SIFB service sequences), indication of different priorities must be provided by IEC 61499 tools in order to avoid confusion of users during design, development, and testing stages. Users shall be allowed to modify priorities (orders) in IEC 61499 tools for deterministic.

Secondly, as events are no longer pure notification, the data structure embedded in an event could be used in BFB or SIFB algorithms. In existing IEC 61499 software design processes, timers must be placed separately outside BFBs and once timer is up, an extra event must be used for notification. Alternatively, timestamp can be passed into function blocks as input variables, but this requires huge manual works. By applying the proposed approach, the initial timestamp and the last execution timestamp of an event could be used as variables directly. Also the current timestamp is available for algorithms by invoking timing services. This will bring benefits for time-related features using IEC 61499.

Finally, the existing approach for the time-stamped discrete-event execution semantic is based on sequential execution: there is always one event being processed at any time in an IEC 61499 resource. Parallel execution provides better performance by dividing event chains into multiple concurrent threads that utilizes maximum hardware resources. However, from the determinism perspective, there are two issues: first, the proposed event queue is a 1-D array, which cannot hold two or more simultaneous events; second, it is impossible that all concurrent threads could complete execution at same time. Varying in execution time of multiple threads will cause non-determinism. For event queues, increasing dimension from 1-D array to 2-D array will be a feasible solution. For the non-determinism issue, one feasible solution is to force other threads to wait until all threads terminate execution. However, the compensation to this solution is again performance.

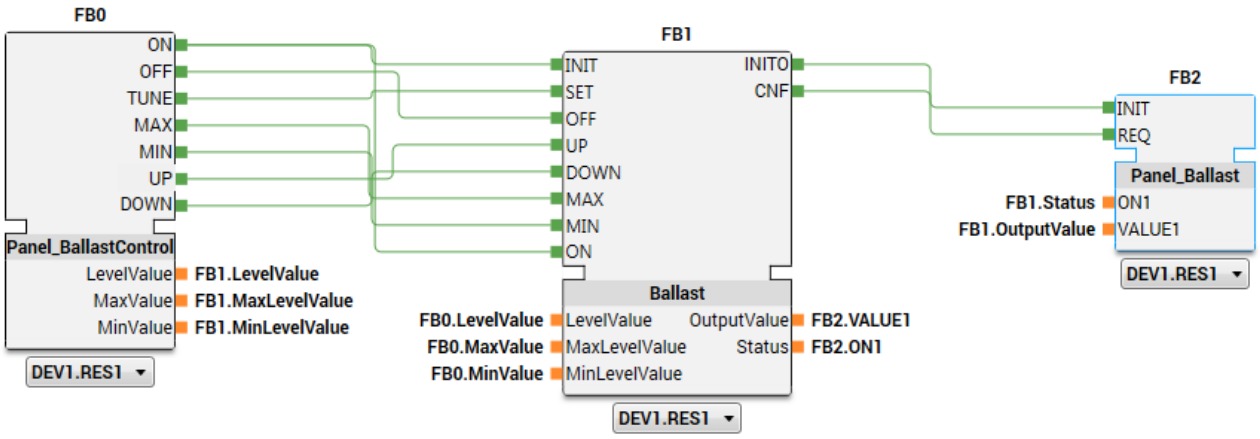

Fig. 6: Lighting Subsystem: IEC 61499 System Design.

VIII. Conclusions And Future Work

One typical characteristic for industrial CPS is diversity: various PLCs cooperate with each other via networks to provide control for complex industrial processes. A time-stamped discrete-event based IEC 61499 execution semantics is proposed for covering real-time constraints and deterministic execution behaviours in industrial CPS execution. The proposed execution semantics embeds timestamps with discrete-event systems and schedules function block network execution by chronological order. The deterministic execution is guaranteed by time-based event handling mechanism defined in the semantic rules.

For the future work, the time-stamped discrete-event based model of computation will be experimented with parallel execution semantics. How to measure real-time constraints needs to be investigated. Furthermore, performance analysis for the proposed execution semantics needs to be performed to compare with other execution semantics.

References

[1] R. Baheti and H. Gill, "Cyber-Physical Systems," IEEE Control Systems Society, 2011.

[2] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems, A Cyber-Physical Systems Approach*, First ed.: LeeSeshia.org, 2011.

[3] V. Vyatkin, "Software Engineering in Industrial Automation: State-of-the-Art Review," *IEEE Transactions on Industrial Informatics,* vol. 9(3), pp. 1234-1249, 2013.

[4] A. Zoitl and H. Prahofer, "Guidelines and Patterns for Building Hierarchical Automation Solutions in the IEC 61499 Modeling Language," *IEEE Transactions on Industrial Informatics,* vol. PP(99), pp. 1-1, 2012.

[5] P. Pihlanko, S. Sierla, K. Thramboulidis, and M. Viitasalo, "An industrial evaluation of SysML: The case of a nuclear automation modernization project," in *18th IEEE International Conference on Emerging Technologies & Factory Automation (ETFA 2013)*, Cagliari, Italy, 2013, pp. 1-8.

[6] R. Drath, "Let's talk AutomationML What is the effort of AutomationML programming?," in *17th IEEE International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, Krakow, Poland, 2012, pp. 1-8.

[7] J. C. Eidson, E. A. Lee, S. Matic, S. A. Seshia, and Z. Jia, "Distributed Real-Time Software for Cyber-Physical Systems," *Proceedings of the IEEE,* vol. 100(1), pp. 45-59, 2012.

[8] Y. Zhao, J. Liu, and E. A. Lee, "A Programming Model for Time-Synchronized Distributed Real-Time Systems," in *13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS 2007)*, Bellevue, WA, US, 2007, pp. 259-268.

[9] A. Albert, "Comparison of Event-Triggered and Time-Triggered Concepts with Regard to Distributed Control Systems," *Embedded World 2004*, pp. 235-252, 2004.

[10] M. M. H. P. Van Den Heuvel, R. J. Bril, Z. Xiaodi, S. Md Jakaria Abdullah, and D. Isovic, "Limited preemptive scheduling of mixed time-triggered and event-triggered tasks," in *18th IEEE International Conference on Emerging Technologies & Factory Automation (ETFA 2013)*, Cagliari, Italy, 2013, pp. 1-9.

[11] J. J. Scarlett and R. W. Brennan, "Re-evaluating Event-Triggered and Time-Triggered Systems," in *11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2006)*, Prague, Czech Republic, 2006, pp. 655-661.

[12] C. Pang, J. Yan, and V. Vyatkin, "Time-Complemented Event-Driven Architecture for Distributed Automation Systems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. PP(99), pp. 1-1, 2014.

[13] *Programmable controllers — Part 3: Programming languages*, IEC Standard 61131-3, 2013.

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

[15] P. Tata and V. Vyatkin, "Proposing a novel IEC61499 runtime framework implementing the Cyclic Execution semantics," in *7th IEEE International Conference on Industrial Informatics (INDIN 2009)* Cardiff, Wales, UK, 2009, pp. 416-421.

[16] V. Vyatkin and J. Chouinard, "On Comparisons of the ISaGRAF Implementation of IEC 61499 with FBDK and other Implementations," in *6th IEEE International Conference on Industrial Informatics (INDIN 2008)*, Daejeon, Korea, 2008, pp. 289-294.

[17] G. Cengic and K. Akesson, "On Formal Analysis of IEC 61499 Applications, Part B: Execution Semantics," *IEEE Transactions on Industrial Informatics,* vol. 6(2), pp. 145-154, 2010.

[18] A. Zoitl, *Real-Time Execution for IEC 61499*, 2nd ed.: International Society of Automation, 2009.

[19] L. H. Yoong, P. S. Roop, V. Vyatkin, and Z. Salcic, "A Synchronous Approach for IEC 61499 Function Block Implementation," *IEEE Transactions on Computers,* vol. 58(12), pp. 1599-1614, 2009.

[20] *Introduction to Discrete Event Systems*, C. Cassandras and S. Lafortune, Eds., 2nd ed.: Springer US, 2008.

[21] E. A. Lee, "Modeling concurrent real-time processes using discrete events," *Annals of Software Engineering,* vol. 7(1-4), pp. 25-45, 1999.

[22] T. Strasser, A. Zoitl, J. H. Christensen, Su, x, and C. nder, "Design and Execution Issues in IEC 61499 Distributed Automation and Control Systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews,* vol. 41(1), pp. 41-51, 2011.

[23] W. Dai, V. Vyatkin, J. Christensen, and V. Dubinin, "Bridging Service-Oriented Architecture and IEC 61499 for Flexibility and Dynamic Reconfigurability," *IEEE Transactions on Industrial Informatics,* vol. in press, 2015.

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

[25] *Digital addressable lighting interface — Part 101: General requirements — System*, IEC Standard 62386-101, 2014.