Towards IEC 61499 Models of Computation in Ptolemy II

Cheng Pang[1], Wenbin Dai[2], and Valeriy Vyatkin[1,3]
[1]Department of Electrical Engineering and Automation, Aalto University, Finland
[2]Department of Automation, Shanghai Jiao Tong University, China
[3]Department of Computer Science, Electrical and Space Engineering, Luleå University of Technology, Sweden
cheng.pang.phd@ieee.org, w.dai@ieee.org, vyatkin@ieee.org

***Abstract*—The prospects of cyber-physical systems (CPS) have been well disseminated and recognized in diverse industries. In industrial automation domain, continuous research on CPS technologies has been funded strategically and globally. One important research challenge in industrial CPS is the modelling of physical processes in continuous domains connected with control systems in discrete domains. Such co-modelling must leverage state-of-the-art standards and tools for practicality. One feasible combination is the IEC 61499 standard for event-driven control and the Ptolemy II platform for heterogeneous model composition. Furthermore, by implementing the computational models of IEC 61499 in Ptolemy II, the behavioural discrepancies of the same IEC 61499 application under different execution semantics can be analysed. As a foundation work towards these goals, this paper investigates the principles of modelling basic IEC 61499 elements using existing Ptolemy II structures. It is aimed to provide some initial insights for engineering industrial CPS based on IEC 61499 and Ptolemy II.**

Keywords—indutrial cyber-physical systems; hybrid simulation; co-modelling; model of computation; IEC 61499; Ptolemy II

I. INTRODUCTION

Recent advancements in information and communication technologies are leading to the emergence of new intelligent systems that are now conceptualized as *cyber-physical systems* (CPS) [1]. CPS are envisioned as the next primary source of disruptive technologies [2]. The engineering of CPS emphasizes the seamless coupling and coordination between the physical and information worlds. This joint consideration, in the broad sense, differentiates CPS from conventionally engineered systems, which compositionally can also be treated as CPS [3]. By synergizing development of communication, control, and computing technologies with physical dynamics as well as human factors, CPS provide the opportunities to realize a new dimension of integrated intelligence. However, such technology convergence, in turn, significantly increases the complexity of system engineering, as many simplification techniques are no longer applicable. For example, the well-known *separation of concerns* design principle cannot be effectively applied to CPS as design concerns in CPS are too closely interrelated to be handled separately. Therefore, new theories and methodologies for engineering CPS must be developed. This new science of CPS cannot be the simple union of existing technologies from information and physical sciences. On the contrary, it must apply systematically integrated multi-disciplinary knowledge to investigate the properties of and interactions between physical, computing, and communication components using techniques such as co-modelling.

Modelling of CPS is intrinsically heterogeneous, where models of physical dynamics in continuous domains and models of computational processes in discrete domains must be jointly considered. The integration of models from different domains is the pivotal challenge [4]. For example, in industrial automation, closed-loop control systems are typically modelled and analysed in continuous time domain. On the other hand, control logics implemented following discrete event models are not unusual. The modelling principles for both domains are well established but not their combination. As a result, the software realization of a stable control loop proved in continuous domain may present instable behaviour after deploying onto control devices.

This work is largely inspired by the Ptides programming approach [5] based on Ptolemy II [6] targeting the above issues. Ptides provides a coordination language for designing CPS, where both computational and physical processes can be modelled in the same framework without knowing hardware details. Ptides is able to explicitly model temporal behaviours of system components, such as computation and communication delays in controllers, sensors, actuators, and networks. From Ptides models, platform-specific executables can be generated with the guaranteed temporal behaviour. While Ptides focuses more on embedded systems, this work attempts to bridge it to industrial automation platforms. In particular, the IEC 61499 control architecture [7] and the Ptolemy II modelling framework are combined. IEC 61499 defines a reference event-driven control architecture for distributed industrial automation systems, which addresses many engineering challenges related to CPS [8]. Ptolemy II supports heterogeneous modelling of diverse semantic domains known as *models of computation* (MoCs). These MoCs and their compositions have rigorous formal foundations [9]. Therefore, by porting the execution semantics of IEC 61499 as MoCs in Ptolemy II, it would be possible to co-simulate distributed event-driven control applications connected with continuous-time plant models with explicitly defined execution semantics for the entire system.

However, the execution semantics of IEC 61499 is not exhaustively defined [10], which results in possibly different behaviour of a control application under different IEC 61499-compliant implementations. Obviously, this is contrary to the standard's original intention of portability, but different IEC 61499 execution models do show their practicality in specific application domains [11]. It is hence beneficial to understand the effects of various execution models on the same IEC 61499 application, and, most importantly, this effect must be investigated in terms of the physical system the application controls. This can be achieved by implementing these execution models as MoCs in Ptolemy II, whose formal semantics of heterogeneous model composition provides the basis for later analysis. Moreover, the concept of time-stamped events in Ptolemy II can be leveraged to address the deterministic execution issues of IEC 61499 applications [12] by complementing the event-driven control chain with the notion of time [13]. Towards these goals, this paper, as a foundation, investigates the modelling principles of basic IEC 61499 elements using existing Ptolemy II structures.

The remainder of this paper is structured as follows. Section II first revises the existing execution models of IEC 61499 and the related works on co-modelling with IEC 61499. Then, Section III delves into the modelling details using an intuitive example. Finally, the paper is concluded in Section IV.

II. Related Works

A. Execution Semantics of IEC 61499

In this section we discuss the main characteristics of execution models used in the five most actively developed IEC 61499 tools with reference to the formal semantics they implemented.

FBDK [14] is the first tool developed to demonstrate the concepts of IEC 61499. Its runtime implements an execution model called *non-preemptive multi-threading resource* (NPMTR) [15], where events are handled immediately after their occurrence following a depth-first scheduling mechanism. As the runtime is none pre-emptive, control algorithms must be short to not block other threads. Moreover, feedback event loops must be avoided to prevent starvation situations. A formal description of NPMTR based on the 1st edition of IEC 61499 can be found in [16].

ISaGRAF [17] is the first commercial tool supporting IEC 61499. For backwards compatibility reasons and for the sake of determinism, it adopts the IEC 61131-3 [18] cyclic scan-based execution model. In each scan, function blocks (FBs) in an IEC 61499 applications are invoked sequentially in a fixed order regardless of whether they receive events or not. This may lead to quite different execution results as discussed in [19]. The formal description of IEC 61131-3 cyclic execution semantics can be found in [20-22] .

4DIAC [23] and nxtStudio [24] are based on the FORTE runtime [25], which implements a queue-based execution model. In general, events in FORTE are stored in a first-in-first-out queue. An event is only dispatched if its preceding event in the queue has been handled. As 4DIAC's runtime is now compliant with the standard's 2nd edition while nxtStudio's runtime is still based on the 1st edition, there are several subtle discrepancies between them as exemplified in [10]. A formal description of the sequential execution semantics can be found in [16].

The runtime of BlokIDE [26] implements a synchronous semantics inspired by the Esterel programming language [27]. At the start of each logical instant of time, called tick, a snapshot of the input events and data is taken. Then, the FBs received input events in current tick will be evaluated simultaneously. The newly generated output events and data will be available at the start of next tick to avoid causality loop issues. This model is quite similar to the cyclic scan-based model except that FBs can be executed in parallel and the duration of each scan depends on the length of each tick. The formal description of BlokIDE's synchronous semantics can be found in [11].

B. Co-Modelling with IEC 61499

There exists many commercial frameworks for modelling physical systems, such as MATLAB/Simulink, Modelica, and Apros. Most existing works on co-modelling with IEC 61499 hence leveraged them. For example, Yang et al. [28] proposed a co-simulation framework of distributed control systems for smart grids. Smart grids are modelled in MATLAB/Simulink while control systems are modelled using IEC 61499. A proxy-based mechanism was proposed to coordinate the different execution speeds of the control and physical models. In [29], Thramboulidis and Buda presented a similar co-modelling framework based on IEC 61499 and Modelica. The focus of this work was more on an integrated modelling approach, where SysML was used to provide a holistic view of the whole design. In both works, socket communication was used to transfer data between plant and controller models. Nikula et al. demonstrated another co-simulation environment for district heating networks in [30]. A broker was implemented to synchronize the execution of the IEC 61499 control model and the Apros physical model. In particular, the broker uses OPC UA interface to advance Apros simulation steps. At the end of each simulation step, the broker transfers the outputs of Apros model to the IEC 61499 model via UDP packets. The new control commands, generated by the IEC 61499 model, are then relayed by the broker to trigger the next Apros simulation step. As different to the majority of the above-mentioned tools, Ptolemy II enables modelling of heterogeneous CPS with well-defined system-level semantics. This is the main reason for adopting this approach in this paper.

III. Modelling of IEC 61499 in Ptolemy II

A. Ptolemy II Actor Semantics

Ptolemy II pursues an actor-oriented design paradigm [31] for conceptualizing and structuring system models. In general, each actor represents a system component. The composition, execution, and communication of actors are governed by sets of rules termed MoCs. The same actor model under different MoCs could behave quite differently. Ptolemy II supports a rich set of MoCs, such as *synchronous reactive*, *discrete event*, *continuous time*, and *modal models*, for different semantic domains. Each MoC implements the actor abstract semantics [32], which establishes the formal foundations for hierarchical

compositions of heterogeneous MoCs. Along with the development of Ptolemy, its actor semantics has also evolved. This work leverages the latest Ptolemy II's formal semantics [9], where a unified framework has been proposed to formally define various MoCs. The basics of actor semantics are briefly reproduced and adapted below.

Definition 1. Variables and valuation:

Let V be a set of variables. A valuation over V is defined as $x: V \to \mathcal{U}$, where $\mathcal{U}$ is the set of all possible values. The set of all possible assignments over V is indicated as $\hat{V}$. In Ptolemy II, there are two special values in $\mathcal{U}$:

- $\perp$ represents the variable's value is unknown; and
- $absent$ represents the value is absent.

Definition 2. An *Actor*, a, is defined as a 8-tuple:

$$a = \langle I, O, S, s_0, f, p, d, t \rangle$$

where:

- I is a set of input variables;
- O is a set of output variables;
- S is a set of state variables;
- s_0 is the initial state, $s_0 \in S$;
- f is a *fire* function: $f: \hat{S} \times \hat{I} \to \hat{O}$;
- p is a *postfire* function: $p: \hat{S} \times \hat{I} \to \hat{S}$;
- d is a *deadline* function: $d: \hat{S} \times \hat{I} \to \mathbb{R}_+^\infty$; and
- t is a *time-update* function: $t: \hat{S} \times \hat{I} \times \mathbb{R}_+ \to \hat{S}$.

The actor abstract semantics specifies the execution of an actor as follows. At state s_i, an actor a is fired with input $x_i \subseteq \hat{I}$. Firstly, the fire function f is invoked to produce output $y_i \subseteq \hat{O}$. Then, the state of a is temporarily updated to s_i' by its postfire function p. The actor computes its deadline for next firing using the deadline function d. At last, the environment (i.e. a MoC) decides the amount of time, to advance. According to this time, a updates its state to s_{i+1}, by calling its time-update function, t. Depending on the implementations of these four functions, an actor can model different behaviours. For example, if d always return infinity, then a is untimed. If t always returns the same state as p, i.e. $s_{i+1} = s_i'$, then a is delay independent.

In this work, two types of actors in Ptolemy II are of interest: *extended state machines* (ESM) and Modal Models (MM). An ESM can be naturally modelled by Definition 2. In particular, S corresponds to locations of an ESM; f and p are defined by the rules of transition evaluation. ESMs are delay independent and untimed Mealy machines. A MM is a network of actors.

Definition 3. *A Modal Model*, M, is defined as a set of actors:

$$M = \{a_c, AR\}$$

where:

- a_c is the controller of M, which is an ESM;
- AR is a set of *refinement actors* for states in a_c, which are executed when corresponding states in a_c are entered; and
- All actors in M share the same input and output variables.

B. Basics of IEC 61499

This section only recapitulates the basics of IEC 61499. The more exhaustive formal definitions can be found in [21, 33]. The primary building units in IEC 61499 are basic FBs.

Definition 4. *A Basic Function Block*, fb, is an 7-tuple:

$$fb = \langle EI, EO, DI, DO, \omega_i, \omega_o, ecc \rangle$$

where:

- EI and EO are respective sets of event inputs and outputs;
- DI and DO are respective sets of data inputs and outputs;
- ω_i and ω_o are the respective functions to associate event ports with data ports: $\omega_i: EI \to 2^{DI}$ and $\omega_o: EO \to 2^{DO}$;
- ecc is an *execution control chart* (ECC).

The algorithm invocation sequence of a basic FB is specified by its ECC, which can be seen as an extended Moore machine.

Definition 5. *An Execution Control Chart*, ecc, is a 5-tuple:

$$ecc = \langle ES, es_0, ET, EA, \delta \rangle$$

where:

- ES is a set of *execution control* (EC) states;
- es_0 is the initial state, $es_0 \in ES$;
- ET is a set of EC transitions;
- EA is a set of EC actions; and
- δ is an assignment function: $\delta: S \to 2^{EA}$.

Definition 6. *An Execution Control Action*, ea, is a pair:

$$ea = \langle alg, eo \rangle$$

where:

- alg is an algorithm; and
- eo is an event output, $eo \in EO$.

Definition 7. *An Execution Control Transition*, et, is a 3-tuple:

$$et = \langle ss, \theta, ds \rangle$$

where:

- ss is an initial EC state, $ss \in ES$;
- θ is a guard function that evaluates $\theta: (EI \cup \{1\}) \times 2^{DI} \times 2^{DO} \times 2^{IV} \to \{true, false\}$; and
- ds is a destination EC state, $ds \in ES$.

C. Modelling Details

Both IEC 61499 and Ptolemy II follow the component-based design paradigm. Simple actors and FBs can be connected via input and output ports to form more complex ones. Due to this syntactic similarity, in principle, the modelling of a basic FB in Ptolemy II can be achieved by mapping its structures to the counterparts of a MM actor. The initial syntactic mapping can be defined as follows.

Input: A basic FB, $fb = \langle EI, EO, DI, DO, \omega_i, \omega_o, ecc \rangle$ and an empty MM, $M = \{a_c = \emptyset, AR = \emptyset\}$.
Output: A populated MM, $M = \{a_c, AR\}$.

1: **procedure** $Mapping(fb, M)$
2: Create an empty ESM, $a := \langle I, O, S, s_0, f, p, d, t \rangle$;
3: **foreach** $i \in EI^{fb} \cup EO^{fb}$ **do**
4: **if** $i \in EI^{fb}$ **then**
5: Create an input variable, bi, of type Boolean;

```
        I^a ← I^a ∪ {bi};
      else if i ∈ EO^fb then
        Create an output variable, bo, of type Boolean;
        O^a ← O^a ∪ {bo};
      end if
    end foreach
    // As DI^fb ∪ DO^fb ⊂ Î^a ∪ Ô^a, so direct mappings of data types are possible.
    foreach j ∈ DI^fb ∪ DO^fb do
      if j ∈ DI^fb then
      Create an input variable, di, with the same type as j;
        I^a ← I^a ∪ {di};
      else if j ∈ DO^fb then
      Create an output variable, do, with the same type as j;
        O^a ← O^a ∪ {do};
      end if
    end foreach
    s_0^a ← es_0^ecc;
    foreach es ∈ ES^ecc do
      Create an actor state, s, based on es;
      S^a ← S^a ∪ {s};
    end foreach
    foreach et ∈ ET^ecc do
      Update f^a and p^a, based on et;
    end foreach
    foreach ea ∈ EA^ecc do
      Create a refinement actor, a_r, based on ea;
      AR^M ← AR^M ∪ {a_r};
    end foreach
    a_c^M ← a;
    return M;
  end procedure
```

It can be spotted from the above algorithm that FB events are mapped to Boolean variables in actors. This is because actors communicate with each other via tokens. A token carrying a Boolean value is hence equivalent to an FB event. On the other hand, data variables can be directly mapped. Moreover, as actors always use the latest inputs, the association functions, ω_i, ω_o, are not mapped. This is equivalent to an FB having all its data ports associated with all event ports. Therefore, we consider a simplified FB semantics by using this "all sampled" rule.

In a MM, its ESM defines its execution behaviour. Similarly, in a basic FB, its ECC specifies the invocation sequence of its algorithms. It should be noted that ESMs are Mealy machines while ECCs are Moore machines. Fig. 1 illustrates the semantic mappings between ECC and ESM using the AddSub FB.

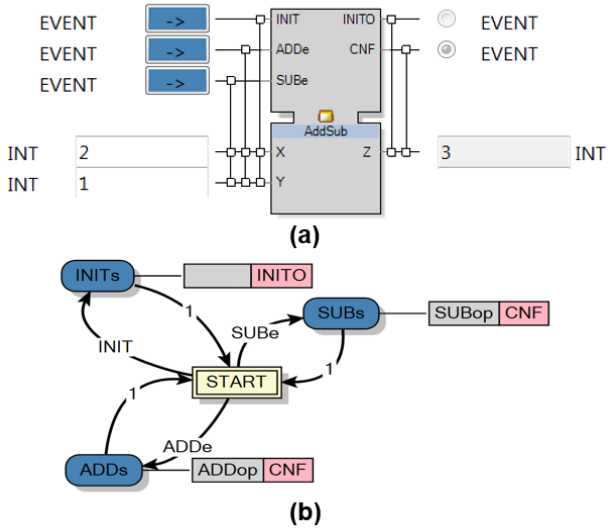

Fig. 1. AddSub FB: (a) Interface in nxtStudio Debug Mode and (b) ECC.

The function of AddSub is as follows: when event ADDe arrives, the value of Z is updated to X+Y. If event SUBe arrives, the value of Z is set to X-Y. The behaviourally equivalent actor model of AddSub is illustrated in Fig. 2. By comparing Fig. 1 and Fig. 2 (b), one can see the clear one-to-one syntactic mappings for ports and states. On the other hand, mapping of transitions is not straightforward. In general, different types of ESM transitions are used to model different ECC operations. There are 3 types of ESM transitions used as listed in TABLE I.

TABLE I. TYPES OF USED ESM TRANSITIONS [6].

Type	Notation	Description
Ordinary Transition	guard: g output: x = y s1 → s2	Fire if g is true; upon transitioning the output variables are set.
Immediate Default Transition	guard: g output: x = y s1 → s2	Fire immediately if g is true and the preceding transition to s1 is also fired. Default transitions have lower priority than ordinary ones.
Transition with Refinement actors	guard: g output: x = y s1 → s2	An ordinary transition with refinement actors, which are fired first before the transition is fired.

According to the *ECC Operation State Machine* defined in the IEC 61499 standard, simple EC transitions without EC action, such as START→INITs in Fig. 1 (b), can be directly mapped to an *ordination ESM transition* by copying the guard conditions. EC transitions with guard condition "1", must be converted into an *immediate default transition* in ESM and setting its guard to TRUE. Moreover, in Ptolemy II, ESM transitions can have only two levels of priority: default and ordinary. In general, ordinary transitions have higher priority

than default transitions. If two ordinary transitions with the same level of priority are enabled simultaneously, Ptolemy II will throw an exception indicating this nondeterministic behaviour. In IEC 61499, the priorities of EC transitions are determined by the order in which they are specified in the XML file. If this order must be preserved in the actor model, additional guards must be introduced. Next, for EC transitions with EC actions, each EC action will be modelled by a refinement actor. As indicated in Fig. 2 (c), the EC action associated with EC state ADDs is modelled by the ADDs_Action actor, which is again a MM actor. The ADDs_Action actor is added to ESM transition START→ADDs instead of the ADDs state as ESMs in Ptolemy II are Mealy machines. The ADDs_Action actor will be executed when the ESM transition START→ADDs is fired. Moreover, as multiple EC actions can be added to an EC state, multiple refinement actors can also be added to an ESM transition. Refinement actors are executed sequentially based on the order in which they are added. This can be used to model the execution order of EC actions, which depends on their orders in the XML file again. Furthermore, simple algorithms involving conditional statements, assignments, etc. can be directly translated into an ESM as exemplified in Fig. 2 (d).

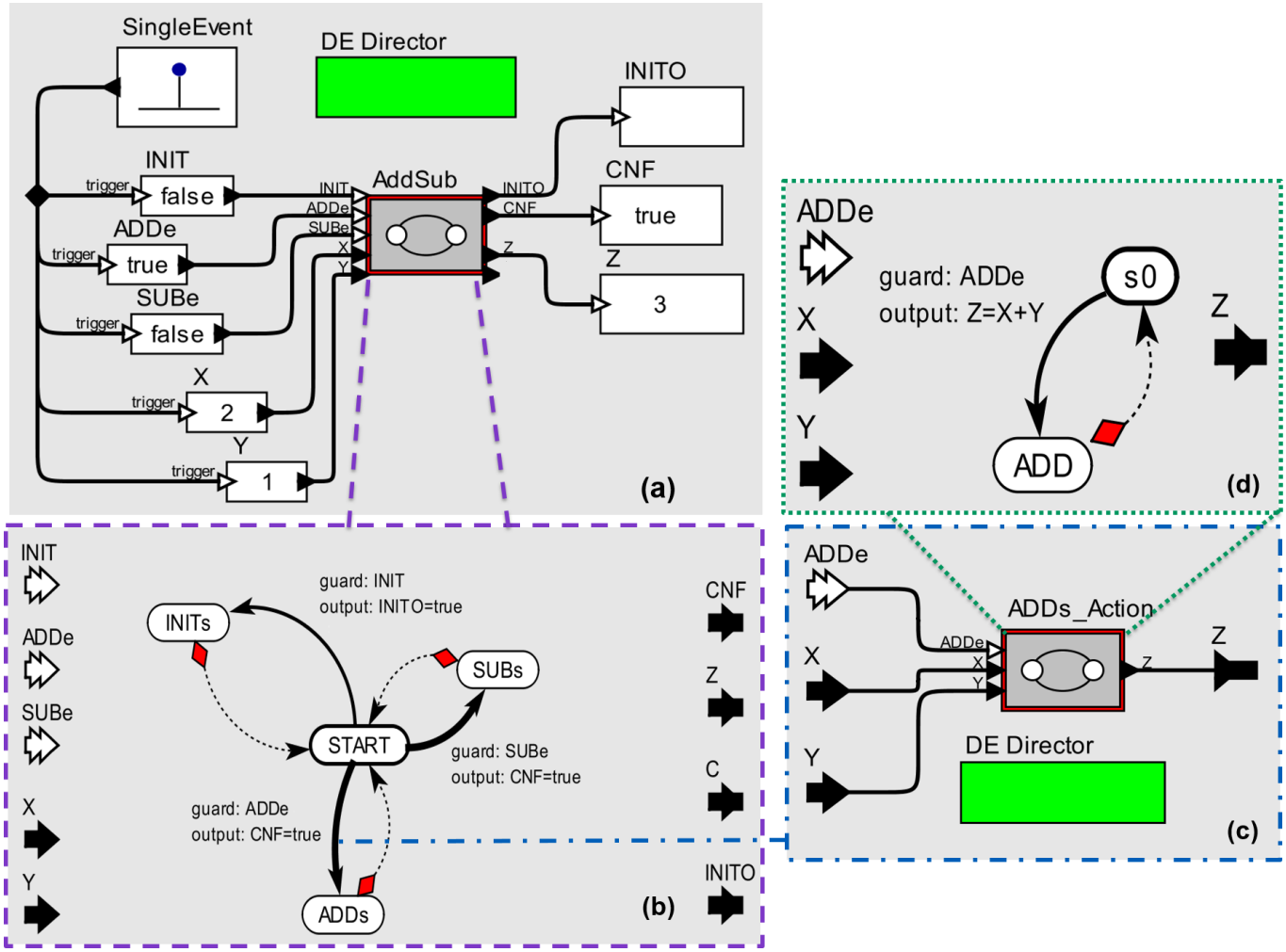

Fig. 2. (a) Interactive Setup, (b) AddSub Actor, (c) START→ADDs Transition Refinement, and (d) ADDs_Action Actor.

Finally, as shown in Fig. 2 (a) an interactive setup has been developed in Ptolemy II to simulate the debug mode in nxtStudio (Fig. 1 (a)). This provides an intuitive way to test the functions of modelled FBs. It has been validated that the actor model of AddSub FB produces the same calculation results.

D. Further Discussion

As previously mentioned in Section III.A, there are a number of predefined MoCs in Ptolemy II. Some of these MoCs have very similar execution semantics as the IEC 61499 runtimes summarized in Section II.A. For instance, both the synchronous reactive (SR) MoC and the BlokIDE runtime follow the principles of synchronous languages. While the SR MoC implements the fixed-point semantics ([6], Chapter 5), BlokIDE is based on the Esterel language ([11], Chapter 3). This commonality is the foundation for porting BlokIDE execution model to Ptolemy II. Another example is the discrete event (DE) MoC and the FORTE runtime. They are both event-driven. However, the DE MoC allows simultaneous processing of events, while events are handled sequentially in FORTE. To map the FORTE runtime to Ptolemy II, an explicit model of event queue must be created in the DE MoC to enforce the sequential dispatching of events. It is also possible to simulate

the cyclic execution semantics of ISaGRAF in DE MoC. This will require an additional clock to generate periodic events.

IV. Conclusion

The intelligence of CPS emerges from seamless integration and interaction between physical and cyber components. New engineering approaches for CPS require the joint consideration of computational and physical behaviours, which are likely based on co-modelling approaches. This paper investigated the possibility of modelling IEC 61499 control applications in the Ptolemy II framework. Specifically, as the foundation work, this paper illustrated the formal semantic mappings between basic IEC 61499 structures and Ptolemy II actor models. An intuitive example has been demonstrated.

In subsequent research, the more exhaustive implementation of various IEC 61499 execution models will be developed in Ptolemy II. In particular, the runtime models of nxtStudio and BlokIDE will be considered first. This will allow systematic analysis and correct-by-construction synthesis of IEC 61499 executable code from Ptolemy II actor models. Moreover, co-modelling of industrial CPS can be achieved in a unified manner. Another ongoing work [13] is to introduce the notion of time-stamped events from Ptolemy II into the IEC 61499 control architecture. It is envisioned that the symbiosis of time-driven and event-driven design paradigms can offer better engineering methodologies with improved system performance. One such desirable design feature is invariance of software behaviour when deployed to different hardware topologies, as illustrated in [34].

Acknowledgment

This work is partially supported by the S-STEP and SAUNA projects.

References

[1] K.-D. Kim and P. R. Kumar, "Cyber-Physical Systems: A Perspective at the Centennial," *Proceedings of the IEEE,* vol. 100, pp. 1287-1308, 2012.

[2] NIST. (2013). *Strategic Vision and Business Drivers for 21st Century Cyber-Physical Systems* [Online]. Available: http://www.nist.gov/el/upload/Exec-Roundtable-SumReport-Final-1-30-13.pdf

[3] C. Pang, V. Vyatkin, and H. Mayer, "Towards Cyber-Physical Approach for Prototyping Indoor Lighting Automation Systems," in *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC 2014)*, San Diego, CA, US, 2014, pp. 3643-3648.

[4] A. Fisher, C. Jacobson, E. Lee, R. Murray, A. Sangiovanni-Vincentelli, and E. Scholte, "Industrial Cyber-Physical Systems – iCyPhy," in *Complex Systems Design & Management*, M. Aiguier, F. Boulanger, D. Krob, and C. Marchal, Eds., ed: Springer International Publishing, 2014, pp. 21-37.

[5] 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, pp. 45-59, 2012.

[6] *System Design, Modeling, and Simulation Using Ptolemy II*: Ptolemy.org, 2014.

[7] IEC Standard 61499-1, "Function blocks — Part 1: Architecture," ed, 2012.

[8] C.-H. Yang, V. Vyatkin, and C. Pang, "Model-Driven Development of Control Software for Distributed Automation: A Survey and an Approach," *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. 44, pp. 292-305, 2014.

[9] S. Tripakis, C. Stergiou, C. Shaver, and E. A. Lee, "A modular formal semantics for Ptolemy," *Mathematical Structures in Computer Science,* vol. 23, pp. 834-881, 2013.

[10] C. Pang, S. Patil, C.-W. Yang, and V. Vyatkin, "A Portability Study of IEC 61499: Semantics and Tools," in *12th IEEE Conference on Industrial Informatics (INDIN 2014)*, Porto Alegre, Brazil, 2014, pp. 440-445.

[11] L. H. Yoong, P. S. Roop, Z. E. Bhatti, and M. M. Y. Kuo, *Model-Driven Design Using IEC 61499*: Springer International Publishing, 2015.

[12] 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, pp. 41-51, 2011.

[13] 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, pp. 1-13, 2014.

[14] Holobloc Inc. (2015, April 30). *FBDK 2.2 - The Function Block Development Kit* [Online]. Available: http://www.holobloc.com/fbdk2/

[15] C. Sünder, A. Zoitl, J. H. Christensen, V. Vyatkin, R. W. Brennan, A. Valentini*, et al.*, "Usability and Interoperability of IEC 61499 based distributed automation systems," in *4th IEEE International Conference on Industrial Informatics (INDIN 2006)*, Singapore, 2006, pp. 31-37.

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

[17] ICS Triplex ISaGRAF. *ISaGRAF Workbench* [Online]. Available: http://www.isagraf.com

[18] IEC Standard 61131-3, "Programmable controllers — Part 3: Programming languages," ed, 2013.

[19] 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.

[20] W. Dai, V. N. Dubinin, and V. Vyatkin, "Migration From PLC to IEC 61499 Using Semantic Web Technologies," *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. 44, pp. 277-291, 2014.

[21] S. Patil, V. Dubinin, C. Pang, and V. Vyatkin, "Neutralizing Semantic Ambiguities of Function Block Architecture by Modeling with ASM," in *Perspectives of System Informatics*. vol. 8974, A. Voronkov and I. Virbitskaite, Eds., ed: Springer Berlin Heidelberg, 2015, pp. 76-91.

[22] 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.

[23] 4DIAC Consortium. (2008). *Framework for Distributed Industrial Automation (4DIAC)* [Online]. Available: http://www.fordiac.org

[24] nxtControl GmbH. (2015). *nxtSTUDIO* [Online]. Available: http://www.nxtcontrol.com/en/engineering/

[25] T. Strasser, M. Rooker, G. Ebenhofer, A. Zoitl, C. Sunder, A. Valentini*, et al.*, "Framework for Distributed Industrial Automation and Control (4DIAC)," in *6th IEEE International Conference on Industrial Informatics (INDIN 2008)*, Daejeon, Korea, 2008, pp. 283-288.

[26] PRETzel. (2015). *BlokIDE* [Online]. Available: http://timeme.io/

[27] G. Berry and G. Gonthier, "The Esterel synchronous programming language: design, semantics, implementation," *Science of Computer Programming,* vol. 19, pp. 87-152, 1992.

[28] C.-H. Yang, G. Zhabelova, C.-W. Yang, and V. Vyatkin, "Cosimulation Environment for Event-Driven Distributed Controls of Smart Grid," *IEEE Transactions on Industrial Informatics,* vol. 9, pp. 1423-1435, 2013.

[29] K. Thramboulidis and A. Buda, "3+1 SysML View Model for IEC61499 Function Block Control Systems," in *8th IEEE International Conference on Industrial Informatics (INDIN 2010)*, Osaka, Japan, 2010, pp. 175-180.

[30] H. Nikula, E. Vesaoja, S. Sierla, T. Karhela, P. G. Flikkema, A. Aikala*, et al.*, "Co-Simulation of a Dynamic Process Simulator And an Event-Based Control System: Case District Heating System," in *19th IEEE*

International Conference on Emerging Technologies and Factory Automation (ETFA 2014), Barcelona, Spain, 2014, pp. 1-7.

[31] E. A. Lee, S. Neuendorffer, and M. J. Wirthlin, "Actor-Oriented Design Of Embedded Hardware and Software Systems," *Journal of Circuits, Systems, and Computers,* vol. 12, pp. 231-260, 2003.

[32] J. Eker, J. W. Janneck, E. A. Lee, L. Jie, L. Xiaojun, J. Ludvig, *et al.*, "Taming Heterogeneity—The Ptolemy Approach," *Proceedings of the IEEE,* vol. 91, pp. 127-144, 2003.

[33] G. Cengic and K. Akesson, "On Formal Analysis of IEC 61499 Applications, Part A: Modeling," *IEEE Transactions on Industrial Informatics,* vol. 6, pp. 136-144, 2010.

[34] V. Vyatkin, C. Pang, and S. Tripakis, "Towards Cyber-Physical Agnosticism by Enhancing IEC 61499 with PTIDES Model of Computations," in *41th Annual Conference of the IEEE Industrial Electronics Society (IECON 2015)*, Yokohama, Japan, 2015, p. submitted.