Article

An Event-Based Programming Model with Geometric Spatial Semantics For Cyber-Physical Production Systems

Guangxi Wan ^{1,2,3,4,*} and Peng Zeng ^{1,2,3,*}

¹ State Key Laboratory of Robotics, Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang 110016, China

² Key Laboratory of Networked Control Systems, Chinese Academy of Sciences, Shenyang 110016, China

³ Institutes for Robotics and Intelligent Manufacturing, Chinese Academy of Sciences, Shenyang 110169, China

⁴ University of Chinese Academy of Sciences, Beijing 100049, China

* Correspondence: wanguaxi@sia.cn (G.W.); zp@sia.cn (P.Z.); Tel.: +86-131-3423-5675 (G.W.)

Received: 29 September 2020; Accepted: 26 October 2020; Published: 29 October 2020

Abstract: With the increasing complexities, such as massive computing devices and strict requirements for collaboration, of industrial production systems, the concept of the cyber-physical production system (CPPS) is considered as a promising approach for addressing these challenges. However, programmability is a challenge in CPPSs. The traditional chimney style of programming requires considerable design effort from engineers to handle the spatiotemporal heterogeneity of cyber-physical model that encompasses computing, temporal, and geometric semantics along with physical dynamics. Thus, an easy-to-use programming model and an integrated programming framework that are suited to this context are required. However, the existing programming models typically fully consider only computing while only partially considering temporal characteristics, and they rarely consider geometric semantics. To solve this problem, this paper proposes a novel event-based programming model, GePro, considering the geometric spatial semantics to realize integrated programming and reduce design effort, especially the reconfiguration of adding new physical devices into the exist system. A prototype of GePro is implemented and verified based on IEC 61499 by the implementation of design and reconfigure assembly CPPS. The results show that using GePro results in a programming time savings of an development life cycle time compared to traditional models for reconfiguration.

Keywords: programming model; CPPS; IEC 61499; function block

1. Introduction

Cyber-physical production systems (CPPS) tightly integrate the entities of cyber and physical spaces, such as scheduling services and physical execution devices, for industrial intelligent production automation [1]. To realize the control and management of a large-scale collection of dynamic heterogeneous resources involving both computational and physical components, a CPPS is endowed with autonomy, collaborative capabilities, efficiency, flexibility, and reconfigurability [2]. However, to implement a CPPS application, engineers must divide it into various parts and program each part separately because the compute devices are likely to be heterogeneous platforms and a CPPS is composed of multiple subsystems (e.g., motion control, sensor perception, and data analysis). Most programming efforts in the context of CPPS have required manual development and careful tuning, and consequently they are not scalable or extensible. This chimney-like programming pattern causes engineers enormous difficulties in programming and system integration. Thus, an easy-to-use

programming model and an integrated programming framework that are suited to this context are required, as shown in Figure 1. The system designer can build CPPS applications directly from software components without additional development effort, which requires that the computing, temporal and geometric characteristics can be inherently handled in software components.

Figure 1. Current and desired programming patterns for cyber-physical production system (CPPS).

One of the stumbling blocks hindering the achievement of integrated programming at embedded devices is the lack of a programming model for CPPS components that considers both computational and physical dynamic processes. The temporal continuity of the physical world is an intrinsic property of CPPS and affects the determinism of system behaviors in response to concurrent events, but the existing computational models do not reflect this temporal property. Likewise, the geometric spatial relationships of the physical objects involved determine the consistency of the behavior of multiple devices collaborating in Cartesian space. For example, consider a case in which two robots are performing assembly tasks in the same work area. Both of them are capable of doing the tasks alone, but when they are placed together in the same area, the outcome becomes uncertain because of the possibility of collisions affecting the physical dynamics, thus imposing both geometric and temporal constraints. This requires additional development work to coordinate them that means the system designer cannot use the software component in a “plug-and-play” way, which violates the intention of the integrated programming framework in Figure 1. Therefore, it is necessary to propose a programming model that can cover the above characteristics of the CPPS model.

The existing programming models consider computing-related issues and partially consider temporal characteristics, but they rarely consider geometric semantics. Additional development by engineers is usually required for the relevant geometric constraints and transformations, which violates the vision of the integrated programming framework.

Concerning the time-related semantics of CPPS components, the PTIDES programming model [3] explicitly models the temporal behavior of system components such as controllers, sensors, and actuators as well as computing and communication delays in the network to guarantee the determinism of system behaviors that is independent of platforms. This characteristic makes a great contribution to the design and analysis of cyber-physical system.

The IEC 61499 standard provides architecture and software models for the next generation of industrial automation systems [4]. The programming model is designed on the basis of event-triggered function blocks (FBs) [5]. The FB concept extends the subroutine structure in IEC 61131-3 [6] to a functional unit in a distributed system, usually in the form of a system-level executable modeling language [7]. FBs provide a unified interface encapsulation for logical code and hide complexity from users through various hierarchical levels. The distributed and event-driven features of IEC 61499 naturally realize the decoupling of different functions and the composability of the system application, which provides a component-based design paradigm for CPPS. To avoid the non-deterministic behaviors caused by the *real-time constraints* and *concurrent events handling* in the distribute system,

the concept of a time-stamped event of PTIDES is adapted into a IEC 61499 discrete event model. However, the geometric spatial semantics are not taken into consideration.

In this paper, we intend to propose a programming model that covers the computational, temporal, and geometric spatial properties of CPPS component to support integrated programming framework mentioned. The main contributions of this article are as follows.

- A programming model that considers geometric semantics is proposed, GePro, which bridges the gap that lacks geometric semantics of the CPPS model and accelerates the system design with software with “plug-and-play”.
- Following the syntax and semantics of the proposed programming model, we give a prototype implementation based on IEC 61499 FB.
- We use an assembly system design process to show how a domain expert uses GePro to design CPPS components, and a reconfiguration process to show how system designer use GePro-based software components to quickly rebuild systems without any extra programming efforts.

The remainder of this paper is organized as follows. Related work on programming models for CPPS is reviewed in Section 2. Section 3 presents the preliminaries necessary for the introduction of GePro. In Section 4, we first give the representation of geometric values in heterogeneous computing devices, which avoid cumulative deviations causing inaccurate geometric information. Second, we redefine the event model and the execution control action considering geometric semantics. Third, the detail execution semantics of the GePro is given. The prototype implementation of GePro-based FB is described in Section 5, as well as the design process by domain experts and use process by system designers. Section 6 summarizes the conclusions of this paper.

2. Related Works

Models of computation (MoCs) for cyber-physical components can be categorized into dataflow models, network models, synchronous-reactive models, finite state machines, discrete event models, time models, etc. Most of them are founded on the separation-of-concerns design principle and consequently focus on only one aspect of the modeling problem, but a few consider both temporal characteristics and geometric information in a synthetic manner.

The time states in a cyber-physical system (CPS) have a dual nature, being both discrete and continuous, and a corresponding mixed temporal description can be realized in the form of hybrid automata [8–10]. Hybrid automata are defined by a directed multivariate graph. Each vertex in the graph is defined as a mode, i.e., a discrete form. The lines connecting the vertices represent modal transformations, which may include transitions from a continuous state to a discrete state. The flow condition corresponding to each discrete point defines a continuously variable feature of the discrete state, which is expressed by means of calculus. However, hybrid automata do not provide a suitable programming model for implementation. Related modeling paradigms, such as Petri nets [11,12] and Modelica [13], are mainly used for analysis and verification.

An actor-oriented programming model has been proposed by the Ptolemy project [14]. The PTIDES model includes a temporal concept of interaction with the environment. It is based on a discrete-time or discrete-event (DE) computing model, in which the software and hardware components, called actors, send messages associated with time-stamped events [15]. The DE paradigm specifies that each actor should process events in time stamp order; therefore, the order in which events are processed is independent of the physical times at which the events are delivered to the participants. Traditionally, the DE approach is used to build simulations; however, in PTIDES, DE models are executable specifications. Nevertheless, as mentioned before, this actor-based programming model lacks any consideration of spatial semantics related to physical geometry.

In the IEC 61499 standard, FBs are not explicitly equivalent to execution semantics. An enhanced FB programming model [16] has been proposed for DE-based deterministic execution semantics with timestamps. This enhanced FB model introduces the timestamped event paradigm proposed

by the Ptolemy project into the DE-based computational model of the original FB concept and considers execution following several semantic rules. However, like the actor-based model, it still lacks consideration of the semantics of the physical geometric space.

Several design and implementation models for industrial automation systems are compared in [17], including VDSL, IEC 61499 and IEC 61131, Petri nets, and their extensions. They are the formal methods currently used for dependability in industrial automation systems, but none of them provide the geometric semantics.

In summary, the programming models described above have contributed greatly to the design of CPPS and serve as the cornerstone for implementing computational CPS components. Although a temporal modeling concept has been developed to enable the handling of concurrent events to ensure deterministic system behaviors, the geometric semantics of physical dynamic processes are still neglected. To address this issue, this paper proposes an FB-based programming model called GePro that considers geometric semantics along with an event-triggered temporal mechanism of IEC 61499.

3. Preliminaries

3.1. The FB-Based Programming Model

The FBs in IEC 61499 are categorized as basic FB (BFB, state machines), service interface FB (SIFB platform dependent implementations), and composite FB (CFB)/subapplication (encapsulating function networks). Our work is based on BFB, so only the BFB model is introduced here.

Definition 1. A basic function block, *bfb*, is defined as an 8-tuple:

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

where *EI* and *EO* are sets of event inputs and outputs, respectively; *DI*, *DO*, and *INvar* are sets of data inputs, data outputs, and internal variables, respectively; ω_i and ω_o are functions for associating events with data, $\omega_i : EI \rightarrow 2^{DI}$ and $\omega_o : EI \rightarrow 2^{DO}$; and *ecc* is the execution control chart, which works in collaboration with the finite-state-machine behavior.

Definition 2. An execution control chart, *ecc*, is defined as a 5-tuple:

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

where *ES* is a set of execution control (EC) states; es_0 is the initial state, $es_0 \in ES$; *ET* and *EA* are sets of EC transitions and actions, respectively; and ρ is an assignment function, $\rho : S \rightarrow 2^{EA}$.

Definition 3. An execution control action, *ea*, is defined as an ordered pair:

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

where *algo* is an algorithm and *eo* is an event output, $eo \in EO$. Note that *eo* can be absent.

Definition 4. An execution control transition, *et*, is defined as a triple:

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

where *ss* is a source EC state, $ss \in ES$; σ is a condition function that calculates $\sigma : (EI \cup \{1\}) \times 2^{DI} \times 2^{DO} \times 2^{INvar} \rightarrow \{true, false\}$; and *ds* is a destination state, $ds \in ES$. To guarantee the determinism of event-triggered systems, a time stamp is introduced to extend the FB-based model in [18], as defined below.

Definition 5. An event model, *e*, is defined as a triple:

$$e = \langle T_{dl}, T_{last}, P \rangle$$

where $e \in EI \cup EO$; T_{dl} is the deadline for the processing of this event, representing the worst-case execution time; T_{last} is the last time this event was updated, used to distinguish the order of events in the queue; and *P* is a priority, used for identifying simultaneous events. More exhaustive specifications can be found in [18].

3.2. Basic Facts Regarding the Physical Geometric Space

3.2.1. Geometric Model

A physical geometric model can be represented in two alternative forms: a boundary representation or a solid representation. For example, a ball can be modeled as a boundary representation through an equation of the form $r^2 = x^2 + y^2 + z^2$, where r is the radius of the ball and x , y , and z are values of the three-dimensional world, which is a 3D Euclidean vector space R^3 , or a solid representation can be formulated as the set of all points that are contained within the sphere. Both alternatives are considered in engineering applications.

From the perspective of robot operations, there are two kinds of entities in the world: (1) obstacles, which are the occupied parts of the world, denoted by the obstacle region $\mathcal{O}$, and (2) robots, which are geometrically modeled entities controlled via motion planning. During a collaborative process, a robot needs to perform motion planning within a certain time window to avoid the obstacle region that is occupied by another robot.

3.2.2. Motion Spaces

The motion of a rigid robot's manipulator or end effector can be decomposed into rotational and translational motion. The origin and coordinate basis vectors of the world will be referred to as the world frame, and any point in the world can be expressed in terms of the world frame. Thus, motion causes transformations of distances, angles, and orientations in reference to the world frame.

Definition 6. A rotation causes the transformation of the attitude of a rigid body in accordance with the special orthogonal group, $SO(3)$:

$$SO(3) := \{R \in \mathbb{R}^{3 \times 3} | \det(R) = 1, RR^T = I\}$$

where I is the three-dimensional identity matrix and R is called a rotation matrix.

Definition 7. Whereas the group $SO(3)$ is used to represent the rotation space, a topological space is established to describe both rotations and translations; this space is called the special Euclidean group, $SE(3)$:

$$SE(3) := \left\{ \begin{pmatrix} R & v \\ 0_{1 \times 3} & 1 \end{pmatrix} \in \mathbb{R}^4 | R \in SO(3), v \in \mathbb{R}^3 \right\}$$

The part of the matrix above denoted by R realizes rotation, and the part denoted by v realizes translation.

4. Event-Based Programming Model With Geometric Spatial Semantics

The event-based programming model with geometric spatial semantics that this paper proposed is an extension of IEC 61499 BFB and abbreviated as GeFB. Given the possibility of multiple state transitions in the CPPS collaboration, *ecc* of BFB is suitable for modeling these states and transitions. GeFBs are the CPPS components that represent the related functions, variables, and control interfaces of physical objects, with the corresponding semantics in terms of time and geometry.

Coupling the logical/behavioural aspects contained within a FB to a spatial entity defined separately is an alternative way for achieving a geometric-considered application, but it still needs to process geometric space variables extra. By extending the semantics of events, geometric information variables can be passed through event-driven mechanisms, as well as time semantic in [16].

The CPPS structure is the composition of GeFBs, which means that the collaboration between physical objects is equivalent to the interaction between the corresponding GeFBs. Therefore, if we consider the physical geometric semantics when developing the GeFBs, then the GeFBs can be directly used in a “plug-and-play” manner. Along with temporal semantics, event identification, time-constrained event scheduling, and execution monitoring are utilized to guarantee the deterministic behavior of the system. To address time-related issues, the above mechanisms are implemented in the following [18]. Therefore, this paper mainly focuses on the geometric semantics. Note that the original syntax and semantics of the FBs can be found in [18,19].

4.1. Numeric Representation of Space

Heterogeneous computing platforms cannot perfectly reproduce identical real numbers due to the differences in the computational accuracy of their processors, and even tiny differences can be amplified through a series of transformations. This paper solves this issue by considering the spatial resolution.

The spatial frame is represented by $\delta = (\delta_x, \delta_y, \delta_z) * \theta$, where δ_x , δ_y , and δ_z are integer numbers and the spatial resolution θ is a double precision floating point number. If all platforms have the same spatial resolution throughout their execution, then arithmetic computations of spatial values will not suffer from quantization errors.

4.2. Syntax

The GeFB is the basic unit of programming, and a CPPS application consists of a network of ECProFBs. Suppose that we have an GeFB library $\mathcal{L}$, where each $\text{GeFB} \in \mathcal{L}$ is an 8-tuple as defined in Definition 1. However, the definitions of *ea* and the event model as given in Definitions 3 and 5, respectively, are expanded as introduced below.

Definition 8. An execution control action is renamed as an action primitive, $ap \in SO(3)$, which is defined as a 4-tuple:

$$ap = \langle g, algo, \pi, eo \rangle$$

where g is a guard function, which checks the constraints before the action primitive is executed; $algo$ is the algorithm for the action primitive; π is a boundary function, which approximates the geometric action space of the physical entity during the time of execution of the action primitive; and eo is an event output, $eo \in EO$.

Suppose that two robots are working on the same workbench, to collaboratively assemble a specific picture with Lego bricks. *Fetch* and *Place* are two action primitives. *Fetch* requires a robot to fetch a Lego brick from the feed zone. *Place* requires a robot to place the Lego brick it is currently holding in a specific position. Here, we consider *Fetch* as an example.

The guard function g for *Fetch* needs to constantly check to see whether there are any obstacles present above the feed zone. Let L , W , and C represent the length, width, and geometric center, respectively, of the rectangular feed zone, where $C = (c_x, c_y, c_z) * \theta$ indicates the position of the feed zone in space. The reachable region above the feed zone is represented by

$$\Gamma = \left\{ (x, y, z) \left| \begin{array}{l} (|x - \theta c_x| = L/2) \wedge \\ (|y - \theta c_y| = W/2) \wedge \\ (|z - \theta c_z| = \epsilon) \end{array} \right. \right\}$$

where ϵ is the height above the feed zone.

The guard function is $g : \Gamma \cap \mathcal{O} \rightarrow \{true, false\}$, where $\mathcal{O}$ is the obstacle region, which is also the boundary of the other robot's action space.

The *Fetch* algorithm $algo$ and the boundary function π are executed serially when the guard function g returns a value of “true”. The algorithm $algo$ includes the motion planning stage and the trajectory tracking control stage. The trajectory τ obtained via motion planning is discretized into trajectory segments $\tau_1, \dots, \tau_T$, where τ_T indicates that the *Fetch* primitive is about to be completed. To track the segments τ_t , $t = 1, 2, \dots, T$, the tracking controller applies the constraint space $\mathcal{Q}_t$ to the joints of the robot. The robot's motion space is approximated by π as the trajectory is discretized.

$$W(\tau) = \bigcup_{t=1}^T \{((\psi_x(q_n), \psi_y(q_n), \psi_z(q_n)) | \psi(q_n) \in \mathcal{Q}_t) \mid \mathcal{Q}_t = \{q | q_n^{dt} \leq q_n \leq q_n^{ut}\}$$

where $q_n = [q_1, \dots, q_n]^T$ are the joint variables, n is the number of joints of the robot, and $\psi(q_n)$ represents the length of the space that the robot can be controlled to reach by the variable q_n in a certain

dimension, with q_n^{dt} and q_n^{ut} representing the lower and upper limits, respectively, of the joint motion. These boundary estimates are associated with a time series, and the number of estimates is related to the specifications for collaborative interaction in the specific application scenario. From a global perspective, the action space of one robot is the obstacle region of the other robot, i.e., $W(\tau) \doteq \mathcal{O}$.

Definition 9. An event model with geometric semantics, Ge , is defined as a 4-tuple:

$$e = \langle T_{dl}, T_{last}, Exp, \Delta M, obj, P \rangle$$

where T_{dl} , T_{last} , and P are defined in Definition 7; Exp is the expression for the solid representation of the boundary function π ; ΔM represents the transformation function of the object of operation, obj , relative to $SE(3)$ in the world coordinates (see Equation (1)); and obj can be either a point or a vector.

$$\Delta M = \begin{cases} R(obj) + v(obj), & \text{if } obj \text{ is a point} \\ R(obj), & \text{if } obj \text{ is a vector} \end{cases} \quad (1)$$

For example, consider a case in which two robot arms have the same object of operation, obj . As the two robots consider only their own coordinate systems when programmed separately, it is necessary to consider how the attitude and position of obj should be transformed during cooperative operation. ΔM represents a robot's action primitive, ap , acting on obj rather than the actual physical location of the object. The receiver can estimate the position of obj by running ap virtually. Therefore, if the two robots each calibrate the initial position of obj with respect to the world coordinate system and then transmit only their respective action primitives acting on the target point, then each robot can calculate the position of obj relative to its own coordinate system. Note that the action primitives of both robots are elements in $SE(3)$.

Models in the form of Ge are designed for both EI and EO . The expression Exp for input events (Gei) is a statement that describes the obstacle region $\mathcal{O}$, i.e., $Exp \equiv \Delta M(\mathcal{O})$. In contrast, the expression Exp for output events (Geo) describes the action space $W(\tau)$, i.e., $Exp \equiv W(\tau)$. Note that the spatial properties described by the formula Exp depend on the motion planning algorithm, e.g., C-space motion planning [20].

4.3. Semantic

The GeFB still has event-driven execution semantics but with time and spatial semantics, and the application remain compliant to the event triggered semantics as per IEC 61499. The *ecc* is triggered by event sources [21], and the execution of *ea* algorithms is triggered by the guard function for the time period corresponding to an action primitive.

4.3.1. Modification for Deterministic Execution Semantics with Timestamps

The execution of the FBs can be scheduled in an event queue in accordance with the IEC 61499 resource model within a device model [18]. Given an IEC 61499 resource instance i , the event queue $EQres$ is defined as

$$EQres^i = \bigcup_{n=0}^{nmax} Ge^i \quad (2)$$

where $nmax$ is the size of the event queue and Ge is an event element.

The semantics for deterministic execution with timestamps are defined in [16] by introducing a circular first-in, first-out (FIFO) buffer with a write pointer $wptr$, a read pointer $rptr$, and 8 rules. To support the new event model, this paper modifies the algorithms for DE input and output functions. Algorithms 1 and 2 show the input and output function of event queue $EQres$ for Ge events, respectively.

Algorithm 1 Discrete Event Input Function for GeFB.

```

1: if  $((wptr < rptr) \text{ and } (rptr - wptr > 1)) \text{ or } ((wptr > rptr) \text{ and } (wptr - rptr > 1)) \text{ or } (wptr == rptr)$  then
2:    $EQres[wptr] \leftarrow Gei$ 
3:    $wptr \leftarrow (wptr + 1) \bmod nmax$ 
4: end if

```

Algorithm 2 Discrete Event Output Function for GeFB.

```

1: if  $wptr \neq rptr$  then
2:    $Gei \leftarrow EQres[rptr]$ 
3:    $Gei.Exp \leftarrow Gei.\Delta M(Exp)$ 
4:    $Gei.obj \leftarrow Gei.\Delta M(obj)$ 
5:   call the step function (see [16]),  $rptr \leftarrow (rptr + 1) \bmod nmax$ 
6: end if

```

4.3.2. *ea* Execution Semantic in GeFB

The *ecc* in the GeFB paradigm is executed with the original semantics [19]; however, we redefine the execution semantics of the EA model. For clarity of description, this section elaborates on the robot collaboration scenario that two robots are working on the same workbench.

A execution can be divided into 4 stages: a pre-checking stage, a motion planning stage, an action space approximation stage, and a tracking control stage. Algorithm 3 shows the guard function $g()$, which is executed in the pre-checking stage. The value of $Gei.obj$ is *null* on line 1 of Algorithm 3, which means that the event initiator and event responder do not have a common object of operation. The return value of $g()$ determines whether the event continues to respond. A value of “true” indicates that *obj* is reachable.

Algorithm 4 shows the rest of the stages. In the motion planning stage, represented on line 2 of Algorithm 4, the motion planning algorithm should consider the target position, time constraints, and geometric constraints when generating a trajectory. In the action space approximation stage, the boundary function π approximates the geometric action space $W(\tau)$ with trajectory τ . Meanwhile, the output event Geo is updated. Finally, the tracking control algorithm is executed locally.

Algorithm 3 Guard function $g()$.

Input: $Gei.Exp$ and $Gei.obj$

Output: $\{true, false\}$

```

1: if  $Gei.obj == null$  then
2:   return true
3: else
4:   Calculate reachable region,  $\Gamma$ , with given  $Gei.obj$ .
5:   if  $\Gamma \cap Gei.Exp == \emptyset$  then
6:     return true
7:   else
8:     return false
9:   end if
10: end if

```

Algorithm 4 Action primitive algorithm *ap*.**Input:** *Gei***Output:** *Geo*

```

1: if g.() == true then
2:   motion plan with target : Gei.obj & time constrains : Gei.Tdl & geometric constrains : Gei.Exp
3:   approximate the geometric action space  $W(\tau)$  with boundary function  $\pi$ .
4:   Geo.update() & send Geo
5:   while unfinishedState do
6:     tracking control algorithm ...
7:     stateUpdate()
8:   end while
9: end if

```

5. The Prototype Implementation

In this study, we focused on the geometric properties of the computational model to produce a prototype of the GePro framework. The prototype is based on the 4diac FORTE runtime environment [22], an implementation of the IEC 61499 runtime environment that is an open source framework for distributed industrial automation and control systems.

5.1. The Prototype of GeFB

In this paper, we have not added geometric semantics to the runtime, which is the future work. Equivalently, we consider the above geometric syntax and semantics in the design time to implement the prototype of GePro for verification.

There are different methods to represent the geometric spatial model, such as using a dedicated adapter interface type and putting the data in a dedicated structured type and offering a function library to work on that. They differ in form but are essentially the same. The adapter is designed to reduce the mass connections by allowing that a group of interface elements can connect their own adapter type that decouples the coordination of FBs for independence [23], and the adapter interface is categorized as an event type.

Using the dedicated structured data type and a function library is an alternative implementation method (e.g., the time-stamped event model [16] contains a data structure of current time and priority). The geometric model is implemented inside FB based on a function library and uses a dedicated structured data type that contains the geometric elements that the *Ge* event has to present. This method passes geometric information by binding geometric data to the corresponding event, and we take this way to implement the prototype of *Ge* event.

We choose *Ge* event considering that the geometric spatial information can be updated synchronously with events in distributed system. And there is only one event connection when configuring application.

As shown in Figure 2a, the proposed *Ge* event prototype is implemented by adding additional parameters (i.e., *Exp*, ΔM , and *obj*) to the original event model. When the event output set *EO* is generated, the corresponding parameters in the *Ge* model are also passed along with *EO* to the downstream FB. When the event input set *EI* is triggered, the above parameters are also received by the FB. More details on the delivery of the rest of the parameters of *Ge* can be found in [16]. As 4diac does not use the time stamp event model [16], we use the method of time parameter binding events instead in actual implementation. In this prototype, runtime environments are running on the same computer, so the time is synchronized. The time parameter is the deadline for the processing of the correspond event, representing the worst-case execution time. When the function block generates a *Ge* event, it will output the time parameter of processing the event together. When the downstream function block is triggered by a *Ge* event, it will record the current time stamp.

As mentioned earlier in this paper, two new elements are added to each action primitive, ap : the guard function g and the boundary function π . An EC action, ea , is shown in the left part of Figure 2b. In the action primitive prototype, the state es_1^- is added before es_1 to execute the guard function and an EC transition related to the guard function, $et(g)$, as shown in the right part of Figure 2b. The boundary function acts in parallel to the algorithm that performs the action primitive.

Figure 2. The prototype of GeFB: (a) Ge event and (b) action primitives.

5.2. Implementation Based on Case Studies

Added: In this section, a design process of robot assembly line is used to show how a robot domain expert develop the GeFB, and a reconfiguration process to show the how system designer to build the system with GeFBs in “plug-and-play” way by comparing programming methods that do not contain geometric semantics.

5.2.1. Scenarios

This paper verifies the practicability of GePro through the design stage and reconfiguration stage of the assembly line. The first scenario is an assembly line consists of a robot and two workbenches. Moreover, a two-robot collaborative test scenario by adding a robot to the previous assembly scenario. The assembly task is to hold the bricks together. One robot assembly scenario requires the robot to complete the assembly task alone before reconfiguration, as shown in Figure 3a.

A new robot, Robot 2, is added into the assembly line after reconfiguration. As shown in Figure 3b, each robot can assemble the objects into desired target products on its own, which fetches the object from Table x.1 and places it on Table x.2. Now two robots are needed to collaborate on this assembly task. They fetch the object from Table x.1 and place it on Table 3 in world coordinate system, which means the Tables 1.2 and 2.2 are the same table.

The set-up of our implementation is as follows. Two configuration files for Franka Emika Panda robots, Ubuntu 18.04 with ROS-Melodic, MoveIt with Rviz, 4diac IDE, and FORTE. Revision: The control system architecture has two layers: the GeFB network running in the FORTE runtime environment is to achieve logical control of the robots, and the basic action primitives are executed in Rviz simulator. These two control layers are associated through robot operating system (ROS) framework. The action primitives algorithms to control robot are realized based on MoveIt.

Figure 3. Scenarios of robot assembly line.

5.2.2. Design Stage of Robot Assembly Line

(a) Robot actions

The basic actions of robot are *Fetch* and *Place*. To implement the guard function, a virtual action is defined, named *Check*. All actions are triggered by GeFB via ROS topic protocol. The *Fetch* has two input parameters from GeFB and one from low-level ROS control, as shown in Algorithm 5. The *object*, e.g., box and cylinder, which contains the parameters needed for the operation, like setting *pre-and post-grasp* (line 5–10). *Target* contains the pose information, which informs the robot where and what pose to fetch (line 1–4). The *Place* is similar to *Fetch* except that *Place* use *move_group.place* function at line-15.

The *Check* action performs motion planning to determine whether the target position is reachable, as shown in Algorithm 6. If reachable, it will return a true value. The motion planning process does not control the actual motion of the robot.

Algorithm 5 *Fetch*(*move_group*, *object*, *target*)

```

1: //Setting grasp pose;
2: grasp_pose.header.frame_id ← "panda_link0";
3: pose.orientation ← target.orientation;
4: pose.position ← target.position;
5: //Setting pre-grasp approach;
6: pre_grasp_approach.direction.header.frame_id ← "panda_link0";
7: pre_grasp_approach.params ← object.preParams;
8: //Setting post-grasp retreat;
9: post_grasp_retreat.direction.header.frame_id ← "panda_link0";
10: post_grasp_retreat.params ← object.postParams;
11: openGripper(grasps.pre_grasp_posture);
12: closeGripper(grasps.grasp_posture);
13: //Do fetch
14: move_group.setSupportSurfaceName();
15: move_group.pick(object.name, grasps);

```

Algorithm 6 *Check*(*move_group*, *target*)

```

1: move_group.setPoseTarget(target.pose);
2: bool isReachable ← group.plan();
3: return isReachable;

```

(b) GeFB Design

The implementation of GeFB contains four modules:

- *Ge* event parse module that is used for getting *Exp*, ΔM , and *obj* from the *Ge*, which is equivalent to the semantics of Definition 9.

- Task selection module that is used for selecting task by making a decision based on *obj* and task list, and updating the state of task list.
- Task list module that stores the assembly sequences of product.
- Actions module that contains different action primitives corresponding to different tasks, e.g., action *Check* with parameters *Exp*, ΔM , and *obj*, which regarded as the guard function, *g*, in Definition 8.

The instance of the prototype of GeFB is *Robot1* as shown in Figure 4a, which has two *Ge* events: *PARTNERACTIONEXE* and *OWNACTIONEXE*. The ecc of *robot1* encapsulate geometric information processing and the control logics as shown in Figure 4b. The event parse module and task selection are implemented in *normalExecution* state of ecc. Whenever an external event, *PARTNERACTIONEXE*, triggers FB with time and spatial parameters—*partnerExpParam*, *partnerDeltaMParam*, *partnerObj*, and *partnerTime*—the ecc will turn into the *normalExecution* state for task selection and constraint checking. If a true feedback of constraint checking, the feedback event, *GUARDCHECK_RESPONSE*, will trigger action *Fetch* with parameters of current task. The action *Place* is triggered after finishing *Fetch* and the *OWNACTIONEXE* event will be generated with current behavioral constraint parameters—*ownObj*, *ownExpParam*, *ownDeltaMParam*, and *ownExeTime*—to trigger the downstream FB. Note that the downstream FB can be itself. The GeFB related parameters are instantiated as follows; *ownObj/partnerObj* are instantiated with geometric centers of the manipulated object, *ownExpParam/partnerExpParam* are instantiated with the geometric dimensioning and pose of object, *ownDeltaMParam/partnerDeltaMParam* are instantiated with the value of rotation and translation of an object before and after it is manipulated, and *ownExeTime/partnerTime* are instantiated with the time that executing one action needs.

Figure 4. The GeFB and *ecc* for robot.

The application for robot assembly line can be easily developed by system designer through dragging the GeFB, as shown in Figure 5. The Sub and Pubs are ROS nodes for getting feedback and invoking hardware actions, respectively. The red connection (as shown in Figure 5a and the connection between *PARTNERACTIONEXE* and *OWNACTIONEXE* in Figure 5b represent *Ge* event that transfers the geometric information to itself.

Figure 5. The application for one robot assembly line.

5.2.3. Reconfiguration stage of robot assembly line

When the assembly line needs to add a robot to form a collaborative system, the reconfiguration processes of system designer of using different programming models are as follows.

(a) Reconfiguration without GeFB

The system designer cannot directly use the off-the-shelf FB or a software component that lacks the geometric semantics to build new collaborate assembly system without a secondary development process. To ensure that the actions of two robot executed are compatible with geometry of the physical components and no collision, system designers have to employ one or more domain experts to build software components to rebuild a collaborative assembly application.

The collaborate software components or subsystem is essential to coordinate the actions of two robots, which contains the following functions.

- Scheduling that consider the temporal and geometric spatial constraints to decide which robot should be activated.
- Collision avoiding that take the great effort to plan the trajectories of robots to avoid collision.
- Spatial conversion that converts the coordinate system into the one used by the robot.

(b) Reconfiguration with GeFB

When the assembly line needs to add a robot to form a collaborative system, the reconfiguration of the application only needs to add the GeFB corresponding to the new robot in the original FB network without consider the collision issue.

In this scenario, the system schematic diagram of two robot collaborative assembly system as shown in Figure 6. The *Ge* event between two robots is connected by the red event connections.

Figure 6. System schematic diagram of two robot collaborative assembly system. More details see video at <https://pan.cstcloud.cn/s/SyC6t62uRWM>.

5.2.4. Discussion

Although in the design stage, the workload of design GePro-based software component is equivalent to that of the traditional one, it can reduce a lot of effort in the reconfiguration stage. Compared with the traditional reconfiguration, which requires redesigning the scheduling and modifying the logical relations between components, the reconfigurable applications using GeFB reduce the design effort required for reconfiguration in programming, because the time and spatial constraint problems is solved inside the GeFB and more add-in algorithms and framework enable FB to execute both autonomously and collaboratively.

The ease-of-use of the GeFo-based Programming model is difficult to quantitatively evaluate, so we make a qualitative analysis. In simulation scenarios, the GeFBs are well-encapsulated and store in FB library, which means that system designer can use them as system required by dragging and dropping. This programming paradigm promotes the integrated development framework of CPPS. By contrast, in traditional programming models, a secondary development that the system designer cannot solve it alone is required for reconfiguration and a new development life cycle time is consumed.

An actual two-robot collaboration scenario which we have made without GePro can be found at <https://pan.cstcloud.cn/s/XhRgnCNS9g> and it is very similar to the experimental scenario in this paper. This paper focuses on programming model rather than system performance, so it is reasonable to make a qualitative comparison of the development process as shown in Table 1. The actual collaboration scenario requires extra development life cycle for domain experts (PLC programmer and robot programmer) to deal with the scheduling, spatial conversion, and collision avoiding problems. However, the system designer using GePro-based software component only needs to drag component and draw connections between *Ge* event.

Table 1. Comparison of different programming models.

Item	Extra Development for Collaboration	Extra Design Effort
Scenario without GePro	Scheduling, Spatial conversion, Collision avoiding	A new development life cycle
Scenario with GePro	None, Ge event connections	No extra development

6. Conclusions

The PTIDES model uses the time semantic mechanism to guarantee the deterministic of the execution of the CPS system and can describe the computational and time attributes of CPPS. Similarly, an enhanced FB model [16] has been proposed for the same purpose. However, the geometric property of physical dynamic are ignored.

In this paper, an event-based programming model based on IEC 61499 with geometric spatial semantics, GePro, has been proposed for integrated programming framework of CPPS to bridge the gap of lacking geometric property. The GePro framework considers the representation of geometric information in space and the semantic transformations among different geometric coordinate systems for multirobot coordination. We have defined the syntax and semantics for GePro and have used a prototype operating in a reconfigurable collaborative scenario to verify the practicability of the proposed model in reducing design effort. Adding new robots to the system and performing contact-rich-and-collaborative manipulation actions in Cartesian space requires a new software development life cycle, but the programming diagram with geometric spatial semantics does not need extra development time just drag the correspond software components and connections. The comparison of use cases in this paper shows that GePro enables a programming time savings of an development life cycle time compared to traditional one.

A limitation of this programming model is that it assumes that the robot components have a certain degree of autonomy and can adjust the trajectory or timing of actions according to changes in geometric information. However, the robot programming of the current assembly production line relies heavily on robot teaching methods, which means that the geometric information of its collaborator is

meaningless and the robot controller cannot automatically modify its trajectory responding to changes in geometry. However, advances in robot programming technology will alleviate this limitation in the future.

Author Contributions: Conceptualization and methodology, G.W. and P.Z.; software, validation, formal analysis, investigation, data curation, and writing and original draft preparation, G.W.; resources, supervision, project administration, and funding acquisition, P.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key Research and Development Program of China (2018YFB1700200) and National Natural Science Foundation of China (61903356).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Vogel-Heuser, B.; Hess, D. Guest Editorial Industry 4.0-Prerequisites and Visions. *IEEE Trans. Autom. Sci. Eng.* **2016**, *13*, 411–413. [\[CrossRef\]](#)
2. Rajkumar, R.; Lee, I.; Sha, L.; Stankovic, J. Cyber-physical systems: The next computing revolution. In Proceedings of the Design Automation Conference, Anaheim, CA, USA, 13–18 June 2010; pp. 731–736.
3. Derler, P.; Eidson, J.C.; Goose, S.; Lee, E.A.; Matic, S.; Zimmer, M. Using Pthreads and synchronized clocks to design distributed systems with deterministic system wide timing. In Proceedings of the 2013 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS), Lemgo, Germany, 22–27 September 2013; pp. 41–46.
4. Vyatkin, V. IEC 61499 as Enabler of Distributed and Intelligent Automation: State-of-the-Art Review. *IEEE Trans. Ind. Inform.* **2011**, *7*, 768–781. [\[CrossRef\]](#)
5. *Function Blocks—Part 1: Architecture, IEC61499-1*, 2nd ed.; International Electrotechnical Commission: Geneva, Switzerland, 2012.
6. *Programmable Controllers—Part 3: Programming Languages, IEC 61131-3*, 2nd ed.; International Electrotechnical Commission: Geneva, Switzerland, 2003.
7. Dai, W.; Vyatkin, V. Redesign Distributed PLC Control Systems Using IEC 61499 Function Blocks. *IEEE Trans. Autom. Sci. Eng.* **2012**, *9*, 390–401. [\[CrossRef\]](#)
8. Thiagarajan, P.S.; Yang, S. Modular discrete time approximations of distributed hybrid automata. *Theoret. Comput. Sci.* **2012**, *429*, 292–304. [\[CrossRef\]](#)
9. Kobayashi, K.; Imura, J.-I. Time-sequence based modeling of finite automata in model predictive control of hybrid systems. In Proceedings of the 2007 American Control Conference, New York, NY, USA, 9–13 July 2007; pp. 4186–4191.
10. David, A.; Larsen, K.G.; Legay, A.; Nyman, U.; Wasowski, A. Timed I/O automata: A complete specification theory for real-time systems. In Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control, Stockholm, Sweden, 12–15 April 2010; pp. 91–100.
11. Julvez, J.; di Cairano, S.; Bemporad, A.; Mahulea, C. Event-driven model predictive control of timed hybrid Petri nets. *Int. J. Robust Nonlinear Control* **2014**, *24*, 1724–1742. [\[CrossRef\]](#)
12. Čapkovič, F. Petri nets in discrete-event and hybrid systems modelling, analysing, performance evaluation and control. In *Proceedings of the International Conference Automation*; Springer: Cham, Switzerland, 2017; pp. 3–21.
13. Fritzson, P.; Bounie, P. Modelica—A general object-oriented language for continuous and discrete-event system modeling and simulation. In Proceedings of the 35th Annual Simulation Symposium, San Diego, CA, USA, 14–18 April 2002; pp. 365–380.
14. Ptolemaeus, C. *System Design, Modeling, and Simulation using Ptolemy II*; Ptolemy.org: Berkeley, CA, USA, 2014.
15. Fishman, G.S. Modeling Concepts. In *Discrete-Event Simulation: Modeling, Programming, and Analysis*, 1st ed.; Springer: Berlin/Heidelberg, Germany, 2001; pp. 36–40.
16. Dai, W.; Pang, C.; Vyatkin, V.; Christensen, J.H.; Guan, X. Discrete-Event-Based Deterministic Execution Semantics With Timestamps for Industrial Cyber-Physical Systems. *IEEE Trans. Syst. Man Cybern.-Syst.* **2018**, *50*, 851–862. [\[CrossRef\]](#)
17. Sinha, R.; Patil, S.; Gomes, L.; Vyatkin, V. A Survey of Static Formal Methods for Building Dependable Industrial Automation Systems. *IEEE Trans. Ind. Inform.* **2019**, *15*, 3772–3783. [\[CrossRef\]](#)

18. Cengic, G.; Akesson, K. On Formal Analysis of IEC 61499 Applications, Part B: Execution Semantics. *IEEE Trans. Ind. Inform.* **2010**, *6*, 145–154. [[CrossRef](#)]
19. Cengic, G.; Akesson, K. On Formal Analysis of IEC 61499 Applications, Part A: Modeling. *IEEE Trans. Ind. Inform.* **2010**, *6*, 136–144. [[CrossRef](#)]
20. Yamashita, A.; Arai, T.; Jun, O.; Asama, H. Motion planning of multiple mobile robots for Cooperative manipulation and transportation. *IEEE Trans. Robot. Autom.* **2003**, *19*, 223–237. [[CrossRef](#)]
21. Dai, W.; Vyatkin, V.; Pang, C.; Christensen, J.H. Time-stamped event based execution semantics for industrial cyber-physical systems. In Proceedings of the 2015 IEEE 13th International Conference on Industrial Informatics (INDIN), Cambridge, UK, 22–24 July 2015; pp. 1263–1268.
22. Strasser, T.; Zoitl, A.; Ebenhofer, G. 4DIAC—An open source framework for distributed industrial automation and control systems. In Proceedings of the INFORMATIK, Leipzig, Germany, 27 September–1 October 2010; pp. 435–440.
23. Zoitl, A.; Strasser, T.; Ebenhofer, G. Developing modular reusable IEC 61499 control applications with 4DIAC. In Proceedings of the INDIN 2013, Bochum, Germany, 29–31 July 2013; pp. 358–363.

Publisher’s Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).