*future internet*

MDPI

*Article*

# Development of a Novel Open Control System Implementation Method under Industrial IoT

**Lisi Liu [1,*], Zijie Xu [1] and Xiaobin Qu [2,*]**

[1] College of Mechatronics and Control Engineering, Shenzhen University, Shenzhen 518060, China; 2210294046@email.szu.edu.cn
[2] China Nuclear Power Technology Research Institute Co., Ltd., Shenzhen 518000, China
* Correspondence: liulisi@szu.edu.cn (L.L.); quxiaobin@cgnpc.com.cn (X.Q.)

**Abstract:** The closed architecture of modern control systems impedes them from further development in the environment of the industrial IoT. The open control system is proposed to tackle this issue. Numerous open control prototypes have been proposed, but they do not reach high openness. According to the definition and criteria of open control systems, this paper suggests that the independence between control tasks and the independence between control tasks and infrastructures are the keys to the open control system under the industrial IoT. Through the control domain's formal description and control task virtualization to deal with the keys, this paper proposes a new method to implement open control systems under the industrial IoT. Specifically, given the hybrid characteristic of the control domain, a hierarchical semantic formal based on an extended finite state machine and a dependency network model with the time property is designed to describe the control domain. Considering the infrastructure's heterogeneity in the industrial IoT, a hybrid virtualization approach based on containers and WebAssembly is designed to virtualize control tasks. The proposed open control system implementation method is illustrated by constructing an open computer numerical control demonstration and compared to current open control prototypes.

**Keywords:** open control system; formal description; virtualization; container; WebAssembly

check for updates

**Citation:** Liu, L.; Xu, Z.; Qu, X. Development of a Novel Open Control System Implementation Method under Industrial IoT. *Future Internet* **2024**, *16*, 293. https://doi.org/10.3390/fi16080293

Academic Editor: Stefano Rinaldi

Received: 24 July 2024
Revised: 4 August 2024
Accepted: 13 August 2024
Published: 14 August 2024

**Copyright:** © 2024 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 (https://creativecommons.org/licenses/by/4.0/).

## 1. Introduction

The closed architecture of modern control systems in industrial automation has two meanings: the monolith of control software and the infrastructure dependency of control software, as Figure 1 presents. Generally, users cannot modify the control software's functionality to adapt to production orders or migrate the control software to other platforms to make full use of infrastructures (e.g., computing, storage, networking, operating system, etc.). Not only that, the closed architecture results in a stand-alone operation pattern of control systems. That is, one control system often couples with one actuator tightly. For instance, each machine tool has a computer numerical control system (CNC) as its proprietary brain to control its behaviors, as Figure 1 shows. In this context, users fail to equip actuators dynamically; instead, they need to purchase, install, commission, and maintain multiple similar or even the same control systems to equip actuators. Along with the complexity and scale of control software increasing, modern control systems are more and more closed, impeding them from adapting to new infrastructure paradigms in the industrial IoT to better respond to the mass customization challenge.

An open control architecture is proposed to break the closed architecture of modern control systems. According to the "Technical Committee of Open Systems" of the IEEE, an open system is defined as follows: "An open system provides capabilities that enable properly implemented applications to run on a variety of platforms from multiple vendors, interoperate with other system applications and present a consistent style of interaction with the user". G. Pritschow et al. suggest four criteria to estimate the openness of a control system [1]:

- Portability: Control tasks can be used on different platforms without any changes, while maintaining their capabilities.
- Extendibility: A varying number of control tasks can run on a platform without any conflicts.
- Interoperability: Control tasks work together in a consistent manner and can interchange data in a defined way.
- Scalability: Depending on the requirements of the user, the functionality of the control task and performance and size of the hardware can be adapted.

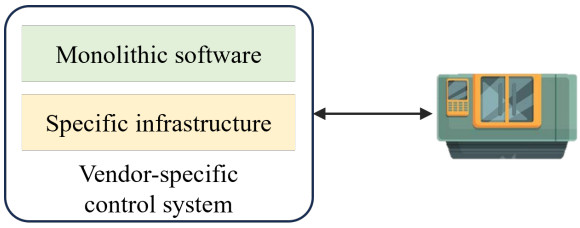



**Figure 1.** The closed architecture of modern control systems.

Over the past three decades, the open control architecture has been discussed extensively, with many open control prototypes developed. A common approach to constructing open control systems is to decompose the control domain into a set of control tasks, implement control tasks as control components according to a certain component model (e.g., CORBA Component Model, Microsoft Component Model, etc.), and then, develop a control prototype by composing control components in a specific structural pattern (e.g., layer architecture, microkernel architecture, etc.) [2–4]. In this case, openness is theoretically implemented by adding, removing, or replacing control components. Actually, however, these component-based open control prototypes do not have a high openness. To be specific, the lack of a control task granularity standard and the lack of a uniform component model to implement control tasks decrease the extensibility and interoperability of control tasks. The widely used off-the-shelf component models are often language- (or runtime-) dependent, which decreases the portability and scalability of control tasks. Moreover, control components may suffer from the "DLL Hell" issue, which occurs when the control prototype at runtime is trying to use an incompatible version of the control component than the one against which it was compiled. Furthermore, control components often deploy on a PC platform or a PC+board platform, making it hard for the component-based open control prototypes to take advantage of the ubiquitous infrastructure resources in the industrial IoT. A detailed review of the component-based open control prototype was presented in the authors' previous work [5].

Compared to the components, services are platform-independent and highly interoperable since they interact via standard messages. Therefore, services are used to develop control tasks. In service-based open control prototypes, control services are often deployed on a cloud platform and publish their functions via exposing the uniform resource locators (URLs) over the network, which formalizes a Control-as-a-Service (CaaS) paradigm [6,7]. But, this faces an urgent challenge: how to satisfy the time determinacy of control systems in the context of the indeterminacy of the network (e.g., latency, packet loss, etc.). A common solution is to distribute control services across cloud and edge platforms in light of their time sensitivity. That is, control services with high time sensitivity are placed on the edge platform, which connects actuators via real-time communication approaches, such as the field bus and real-time Ethernet, while control services with low time sensitivity are deployed on the cloud platform. In this case, openness is implemented by changing the URLs of control services. However, service-based open control prototypes fail to reach a high level of openness as well. In addition to the lack of a control task granularity standard, the reasons attributed to the deployment pattern of control services is relatively static since it is defined in light of the control service's time sensitivity, which is often invariable.

Overall, according to the open system's definition and the openness criteria, the key to implementing open control systems is the independence of control tasks. Here, independence has two meanings: one is the independence among control tasks, and the other is the independence between control tasks and infrastructures. Current open control prototypes attempt to deal with the first independence by defining a standard control task granularity. However, it is hard to define such a granularity standard, especially since the complexity and scale of control software are increasing rapidly. Regarding the second independence, current open control prototypes often develop control tasks in the form of components or services. But, this is incompetent with respect to heterogeneous infrastructures in the environment of the industrial IoT.

This paper, therefore, suggests a new method for implementing open control systems in the environment of the industrial IoT. Specifically, the formal description of the control domain is introduced to cope with the first independence rather than defining a control task granularity standard. Virtualization technology is adopted to isolate control tasks from infrastructures to tackle the second independence. The rest of this paper is organized as follows: Section 2 presents related work about the formal description of the control domain and the virtualization of control tasks; Section 3 overviews the proposed open control system implementation method, with a corresponding demonstration that implements an open CNC system presented in Section 4; Section 5 concludes this paper and presents future work.

## 2. Related Work

### *2.1. Formal Description of Control Domain*

The IEC 61499 standard defines a domain-specific modeling language to construct control systems [8]. Its meta-model is called the function block (FB). As the left side of Figure 2 shows, the FB has separate event and data interfaces. ① When an event input arrives, ② its related data interfaces will be refreshed to obtain data inputs. Then, ③ the event execution control part (top of the FB) executes the input–event–relate algorithms (④ bottom of the FB) and ⑤ generates data outputs and the event output. ⑥ Data outputs refresh the output–event–relate data interfaces, and then, ⑦ the event output is sent. The event execution control part, essentially, is a finite state machine (FSM), as the right side of Figure 2 presents. For instance, the INIT event transits the FB's state from START to INIT. In the state of INIT, the INIT algorithm is executed, and INITO is set as the event output. A control application is made up of multiple FBs via connecting event and data interfaces, which is also called an FB Network (FBN). In addition to the FB and FBN, IEC 61499 also defines the concepts of Resource, Device, and others. Since this paper concentrates on the formal description of the control domain, these concepts are not introduced.

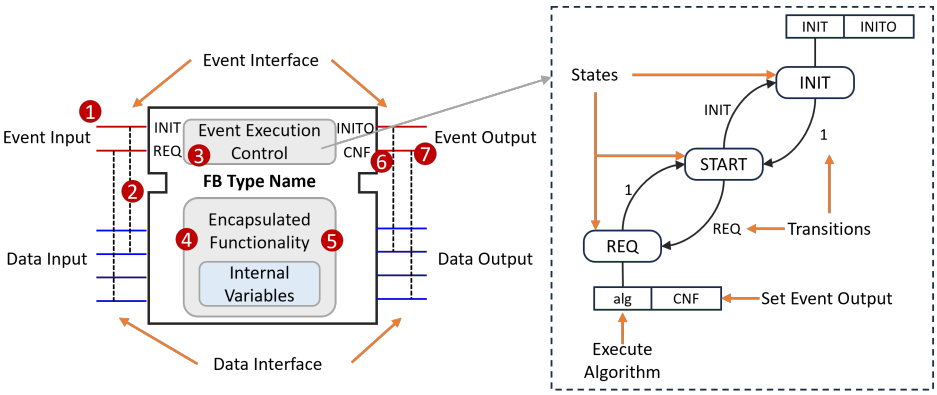



**Figure 2.** Function block of IEC 61499.

It can be seen that the semantic model of IEC 61499 is the FSM, which is a popular model to describe the discrete event system (DES). It conforms to the original intention of IEC 61499, that is an extension of IEC 61131-3 with respect to a vendor-neutral format,

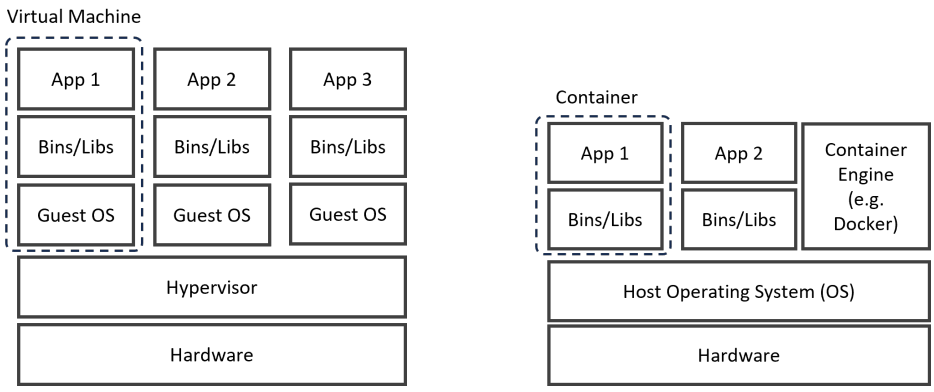
reusability, and controller-to-controller communication [8]. The latter is the international standard for programmable logic controllers (PLCs), which is a typical implementation of the DES. Modern control systems, however, are often a kind of hybrid system that has not only discrete behaviors, but also continuous behaviors. For instance, the CNC is a typical hybrid control system. Its discrete behaviors include I/O behaviors (e.g., spindle rotation, coolant on/off, tool change, etc.), response behaviors to stimuli, and others. Its continuous behaviors mainly include the trajectory control of feed axes in light of a part program. Even though FBs and FBNs are used to develop open control prototypes in some studies [9,10], the FSM is not a proper semantic model to describe the continuous control domain. In other words, despite that the state machine can be the micromodel of software applications, it focuses on the execution of software applications, that is the implementation of software applications rather than the description of the control domain's knowledge.

In addition, the semantics of the FB and FBN are ambiguous [11], for instance the dependency and execution sequence of event inputs are not defined clearly, which leads to the control system described by IEC 61499 not being determined. Many studies have discussed the deficiencies of semantics and proposed corresponding deterministic execution mechanisms [12–16]. However, the execution mechanisms they proposed target particular semantic deficiencies and often rely on particular implementation languages. As a result, there are mutually exclusive IEC 61499 runtime systems that have different semantics.

In summary, with respect to determining control tasks via the control domain's formal description, hey lack of a clear semantic model that can deal with both the discrete and continuous behaviors of control systems.

### *2.2. Virtualization of Control Task*

Regarding virtualization technology, the container attracts more attention than traditional hypervisor-based virtualization in the cloud computing era, thanks to its smaller image size and less boot up time. As Figure 3a shows, hypervisor-based virtualization depends on a hypervisor to abstract hardware resources and monitor virtual machines. A fully functional and bulky guest operating system is needed to run applications. In contrast, as Figure 3b presents, container-based virtualization packages an application with dependencies (e.g., bins, libs, configuration files, and even a base operating system) in a container. A container is a process running on the host operating system that shares the host operating system kernel, while its namespace and control groups are assigned by a container engine to isolate it from others.




(**a**) Hypervisor-based virtualization (**b**) Container-based virtualization

**Figure 3.** Traditional hypervisor- and container-based virtualization.

C. Scordino et al. virtualize the control domain based on Xen, a widely used hypervisor [17]. Three guest operating systems are managed by Xen: a real-time operating system, Windows, and Linux. Each guest operating system is assigned at least one independent core. Real-time and non-real-time control tasks run on the real-time operating system and Windows, respectively, and interact via Linux. To be specific, on the side of Windows,

the Google remote procedure call (gRPC) and protocol buffer cross-platform open-source technologies are used, while on the side of the real-time operating system, remapping to the shared memory and the interrupting service-routine (ISR)-bound reception event communication mechanism are used. It can be seen that this control domain virtualization method does not involve modifying the hypervisor itself. In contrast, RT-Xen is a hierarchical real-time scheduling framework based on Xen [18,19]. The hypervisor layer takes responsibility for instantiating a suite of fixed-priority scheduling algorithms to schedule guest operating systems, while the guest operating system relies on a real-time operating system to schedule tasks. In hypervisor-based control applications, however, the time determinacy performance is heavily influenced by the number of tasks in each guest operating system since the hypervisor fails to allocate hardware resources dynamically to adapt to the guest operating system's demands.

Along with the widespread application of containers in IT infrastructures, many studies try to apply containers in industrial automation [20,21]. Given that a container is actually a running process of the host operating system, it is common to run containerized control applications on a real-time operating system to satisfy the time determinacy. The performance of containerized control applications is mostly estimated by the cyclic behavioral test and the virtual networking behavioral test. The former evaluates the ability to execute application logic at pre-defined intervals and measures accuracy and jitter, while the latter evaluates the ability to communicate between co-located containers in a time-bounded manner. The results indicate that the containerized execution suffers a negligible and constant overhead.

However, current studies on containerized control applications have two main drawbacks: first is the ignorance of the control domain's complexity, and second is the ignorance of the infrastructure's heterogeneity. To be specific, control applications used in current studies are often simple ones for which the whole application is containerized in one container. However, modern control applications are more and more complex and comprise multiple tasks, even distributed control tasks. Among them, not all control tasks have a high time determinacy property, and moreover, the intimacy between control tasks is diverse. Such a complex control application prefers to be containerized into multiple containers according to the control domain's formal description rather than a single one. Moreover, even though the container's image size is smaller than the traditional hypervisor's, as a general-purpose virtualization technique, it is still large for resource-constrained hardware in the industrial IoT. The image size can be further reduced via specialization, that is unikernel-based virtualization [22]. The unikernel aims to build an application-specific operating system in which only the needed libraries are compiled together as a particular kernel [23]. However, currently, the unikernel has two crucial drawbacks: it lacks standardization and requires significant expert work to build, and it has to be rebuilt for every new application.

In summary, with respect to the heterogeneity of infrastructures in the environment of the industrial IoT and the increasing complexity of control software, the container is incompetent in control task virtualization. A potential virtualization method should consider the diverse characteristics and relationships of control tasks.

## 3. Implementation of Open Control Systems Under Industrial IoT

### *3.1. A Hierarchical Semantic Model of Control Domain*

As Section 2.1 mentions, the CNC is a typical hybrid control system in industrial automation; thus, this section takes it as an example to discuss the formal description of the control domain.

On the whole, the CNC's behaviors are mostly discrete ones, which mainly refer to the transitions of a machine tool's states in response to stimuli from users or the environment. Figure 4 presents a simplified state diagram of the discrete behaviors of a machine tool. There are four states, and Idle is the initial state. When a launch event happens, the machine tool state transmits from Idle to Prepared. In the Prepared state, the availability of executing the machine tool's continuous behavior is checked. If available, a success event is generated,

and the machine tool state transmits to the Running state. Otherwise, the machine tool state remains in Prepared until the execution of the machine tool's continuous behavior becomes available or the sleep event happens, which transmits the machine tool state to Idle. In this sense, the priority of the sleep event is higher than the fail event and the success event. In the Running state, the machine tool executes its continuous behavior until a finish event or an error event happens. If the finish event happens, the machine tool state transmits to Idle, while if the error event happens, it transmits to Paused until either the error has been handled and a success event is generated that transmits the machine tool state back to Running, or the error cannot be handled and a fail event is generated, which transmits the machine tool state to Prepared.

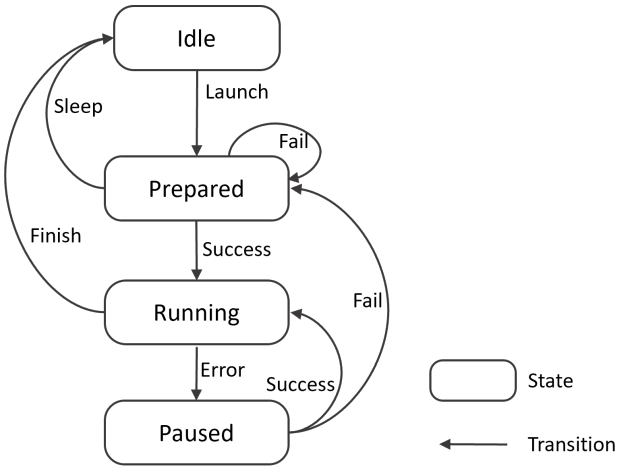


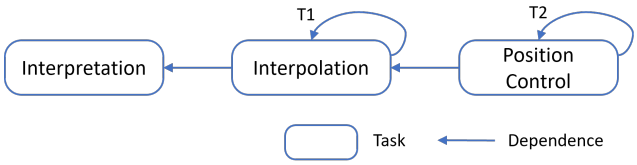

**Figure 4.** State diagram of machine tool's behaviors.

In modern CNCs, in order to control the movement trajectory of feed axes in light of a part program, the part program is generally processed by the following tasks:

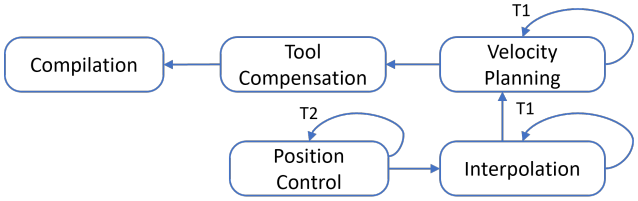
- The interpretation task, which translates the part program into CNC internal commands. In addition to lexical, syntactic, and semantic analysis, computation commands in the part program, such as coordinate transformation, arithmetic expression, tool compensation, and others, should be processed in the translation to obtain the practical tool trajectory.
- The interpolation task, which uses internal commands to generate each feed axis's position data (and velocity data) of each interpolation period in the constraints of the machine tool's kinematics. It often executes cyclically to respond to the change in the machine tool's kinematics.
- The position control task, which generally executes cyclically in each position control period to minimize the position error between the commanded position and the real position feedback by the sensors.

It can be seen that the inputs of the interpolation task are the outputs of the interpretation task, and the outputs of the interpolation task are the inputs of the position control task. In other words, the interpolation task depends on the interpretation task, and the position control task depends on the interpolation task. In this sense, the continuous control domain can be described based on a dependency network model with time property, as Figure 5a presents. The position control task has a dependence on the interpolation task, and the interpolation task has a dependence on the interpretation task. The dependence relationship is transitive. Thus, the availability of the position control task depends on the availability of the interpolation task and the interpretation task. In Figure 5a, it can be seen that the interpolation task has a self-dependence labeled $T_1$, which means it executes every $T_1$ period. This implies a specification that the execution time of the interpolation task cannot exceed $T_1$ and so does the position control task. This implies another specification that $T_1 \geq T_2$ since the position control task depends on the interpolation task.

Moreover, this paper does not aim to define a control task granularity standard; instead, it mainly concentrates on the relationship and intimacy between control tasks. In other words, the control task's granularity size is not fixed, but can be adjusted according to the requirements. For instance, as Figure 5b shows, the interpretation task is divided into a compilation task and a tool compensation task. The former mainly deals with lexical, syntactic, and semantic analysis, while the latter processes the computation commands in the part program. The latter task depends on the former. Similarly, the interpolation task is decomposed into a velocity planning task and an interpretation task, both having a self-dependence labeled $T_1$.




(**a**) Coarse granularity




(**b**) Fine granularity

**Figure 5.** Continuous control domain described by dependency network model.

Based on the above discussion, therefore, this paper suggests a hierarchical semantic model in light of an extended FSM and a dependency network model with the time property to describe the control domain. Figure 6 illustrates this hierarchical semantic model from an object-oriented paradigm perspective. To be specific, there are seven classes: ControlSystem, State, Transition, Event, Action, Task, and Dependence. Here, Action is the abstraction of continuous behavior. As for the relationship between them, a ControlSystem object associates with a State object as its initial state. A State object associates with zero or more Transition objects. Each Transition object associates with two State objects, which indicate its source state and target state, respectively. It associates with an Event object since a transition is triggered by an event. It is noted that the Transition class has a priority property. Moreover, a State object associates with zero or more Action objects, which represents the corresponding continuous behavior that will be executed when the State object is activated by a specific Transition object. Each Action object associates with a Task object as its start task. A Task object associates with zero or more Dependence objects, while each Dependence object associates with two Task objects as its source task and target task. In addition, the Dependence class has a time property and a priority property.

From a mathematical perspective, based on this hierarchical semantic model, a control system can be described by the following septuple:

$$C = (Q, q_0, \Sigma, \delta, F, P_\Sigma, \alpha) \tag{1}$$

where

$Q$: the set of states;
$q_0$: the initial state;
$\Sigma$: the set of events;
$\delta$: state transition function that satisfies $\delta : Q \times \Sigma \to Q$;
$F$: the set of terminal states, which can be empty;
$P_\Sigma$: transition priority function that satisfies $P_\Sigma : Q \times \Sigma \to \mathbb{N}$;

$\alpha$: state action function that satisfies $\alpha : Q \rightarrow Actions$.
Action is described by the following quintuple:

$$Action = (T, t_0, \gamma, P_\gamma, \tau) \tag{2}$$

where
$T$: the set of tasks;
$t_0$: the start task;
$\gamma$: task dependence function that satisfies $\gamma : T \rightarrow T$;
$P_\gamma$: dependence priority function that satisfies $P_\gamma : T \rightarrow \mathbb{N}$;
$\tau$: dependence time function that satisfies $\tau : Q \rightarrow \mathbb{R}$.

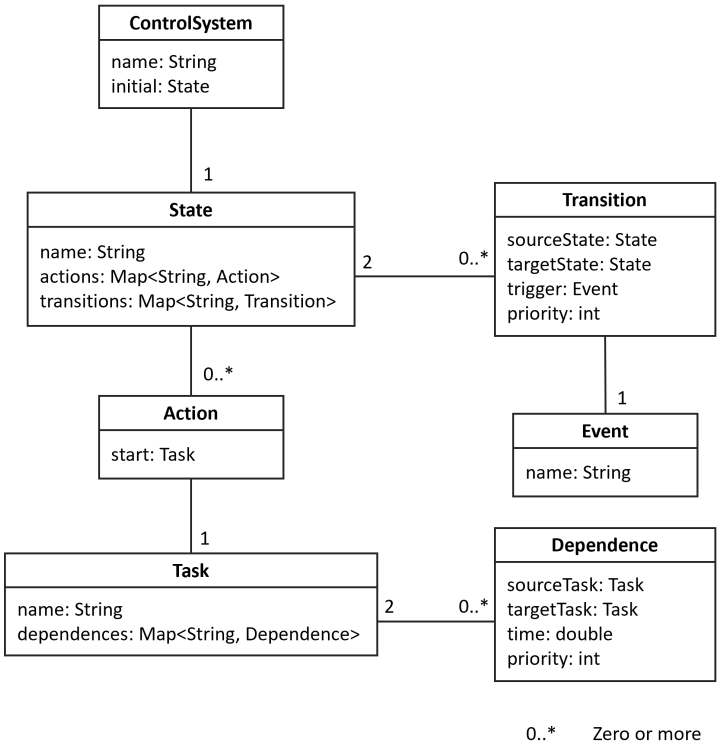



**Figure 6.** Classdiagram of the proposed hierarchical formal model.

### *3.2. Container- and WebAssembly-Based Control Task Virtualization*

As Section 2.2 mentions, container is incompetent with respect to the complexity of the control domain and the heterogeneity of infrastructures in the environment of the industrial IoT. Thus, this paper utilizes WebAssembly as a complementary method to virtualize control tasks.

WebAssembly (wasm) is a compact binary bytecode format released by the major browser vendors to complement and work alongside JavaScript, the only built-in language of the web. It supports browsers to run native code fast, portably, and securely for sophisticated and interactive applications, like 3D games, virtual and augmented reality, and so on [24]. Rather than be written by hand, wasm is the compilation target for multiple source languages, such as C, C++, Rust, etc. That is, codes written in these languages can be compiled as wasm modules. Since wasm is designed with formal semantics from the start, there is no need for an extra trust mechanism to execute wasm modules efficiently and securely [25]. As Figure 7 presents, moreover, with the wasm system interface (WASI) released to abstract the host operating system, wasm modules started running outside of the browser runtime [26]. WASI defines a standardized set of interfaces and functionalities of the operating system in a modular, independent, and capability-oriented manner. The wasm runtime translates WASI calls into the surrounding system calls to finally interact with the host operating system. The wasm runtime provides each wasm module with an

independent sandboxed environment. In a nutshell, wasm is a virtual instruction set architecture (ISA) between the real assembly and Java bytecode (or .NET intermediate language), and its execution depends on WASI to provide a uniform abstraction of different host operating systems. Its advantages, such as its tiny size, low memory footprint, fast booting, great isolation and security, high portability, etc., make it an attractive virtualization technology, especially for resource-constrained hardware in the industrial IoT [27].

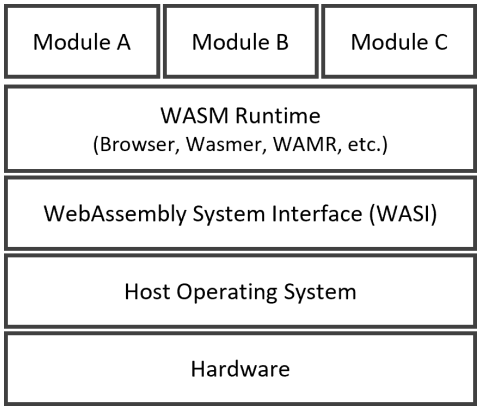



**Figure 7.** WebAssembly-based virtualization.

As an operating-system-level virtualization, wasm-based control applications can also rely on a real-time operating system to satisfy the time determinacy of control systems. In order to evaluate the performance of wasm-based virtualization, two tests have been designed. The first test evaluates the execution efficiency of wasm modules. A bubble sort program written in the C language is compiled into two wasm modules, one in the interpreter mode and the other in the Ahead-of-Time (AoT) mode. The test platform is Raspberry Pi 4B (processor clocked at 1.8GHz, 8G of SDRAM). The result indicates that the average running time of the interpreter mode wasm module is 3.88-times that of the native program, while the AoT mode is only 1.06, near the native program. The second test tries to compile a motor controller into a wasm module. The motor controller contains three real-time threads and a main thread. One real-time thread executes the PID algorithm cyclically, and the other two real-time threads listen to the edge-triggered interrupt of the encode. The main thread initializes the controller and interacts with users. However, the result indicates that the whole controller fails to be compiled as a wasm module because some necessary APIs used in the real-time threads are not defined in the current WASI standard, such as sched_setscheduler(), poll(), signal(), setitimer(), etc. But, the parts of the motor controller that do not rely on these APIs, for instance the user interaction part of the main thread and the PID computation part, can be compiled as wasm modules and have good portability across infrastructures (different operating systems).

In this paper, therefore, a hybrid control task virtualization method based on containers and wasm is designed, as Figure 8 shows. To be specific, control tasks with high time sensitivity are virtualized by wasm and deployed on edge platforms near actuators (i.e., actuator-level edge platforms). Here, as Figure 8 presents, a real-time operating system is often needed to guarantee high time sensitivity. Parts of the control tasks that do not rely on real-time APIs or I/O APIs (such as mostly compute-intensive tasks) are compiled as wasm modules, while other parts are developed as the host application. The dependence between wasm modules and the host application is implemented via shared memory. In contrast, control tasks that do not have high time sensitivity are virtualized by containers and deployed on cloud platforms or high-level edge platforms (e.g., factory-level edge platforms). The dependence between these containerized control tasks and between containerized control tasks and wasm control tasks is implemented by remote communication over the network, localhost communication, shared files, or others, depending on the dependence's intimacy.

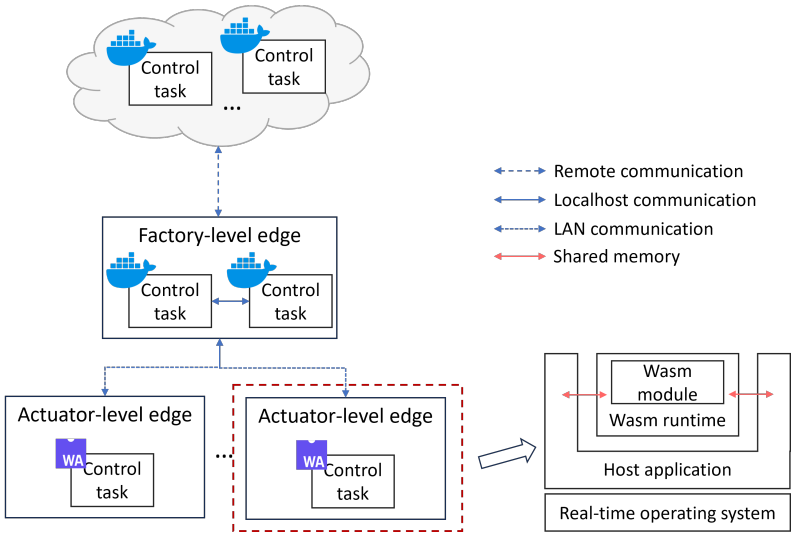
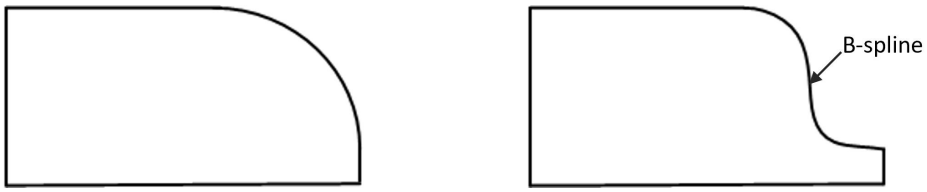



**Figure 8.** Hybrid control task virtualization method.

## 4. Case Study

In order to illustrate the proposed open control system implementation method overall, this section presents an open CNC system demonstration. Generally, traditional CNCs only support linear interpolation and circular interpolation, which means they can directly process the tool trajectory made up of line and arc blocks, as Figure 9a shows. For a complex tool trajectory that contains a B-spline block, as Figure 9b shows, traditional CNCs fail to process it directly, and what is more, they often do not allow the extension of third-party control tasks to process the complex tool trajectory. Instead, such a complex tool trajectory needs to be firstly discretized into a series of line and arc blocks by computer-aided manufacturing (CAM) software. In this context, this section uses the proposed implementation method to design an open CNC system whose functionality can be modified dynamically in response to different tool trajectories.




(**a**) Trajectory consists of line and arc blocks (**b**) Trajectory contains a B-spline block

**Figure 9.** Trajectory example.

### *4.1. The Implementation of an Open CNC Demonstration*

Figure 10 describes the open CNC demonstration's control domain according to the proposed hierarchical semantic model. Here, only two states are presented in order to simplify the figure. Specifically, the Prepared state associates with an action labeled check. When a Launch event switches the machine tool state to Prepared, the check action is activated. The start task of the check action is the Position Control task. The Position Control task associates with two dependences: dependence on the Interpolation task and self-dependence with the time property $T_2$ valued at 2 ms. The former's priority (1) is higher than the latter's (0), which means the Interpolation task should be checked before the Position Control task. Besides, the Interpolation task associates with three dependences: one on the Interpretation task, one on the Look-ahead Velocity-Planning task, and one on itself with the time property $T_1$ valued at 4 ms. Among them, the priority of self-dependence is the lowest (0). It can be seen that, due to the transitivity of the dependence, the dependence on the Interpretation task in Figure 10 is presented in a dashed line since the Look-ahead Velocity-Planning task associates with the dependence on the Interpretation

task as well. Therefore, the check action's sequence is Interpretation -> Look-ahead Velocity Planning -> Interpolation -> Position Control. In addition to the check action, the Prepared state associates with three transitions labeled Sleep, Success, and Fail events, respectively. The Sleep transition has a higher priority (1) than the other two (0). When the check action is performed, a Success event is generated and the machine tool state switches to Running. The Running state associates with two transitions labeled with Finish and Error events. They have the same priority since the Finish and Error events are mutually exclusive. Besides, the Running state associates with the same action as the Prepared state, but is labeled with invoke. A textual representation of this open CNC demonstration is presented in Appendix A in the form of a configuration file.

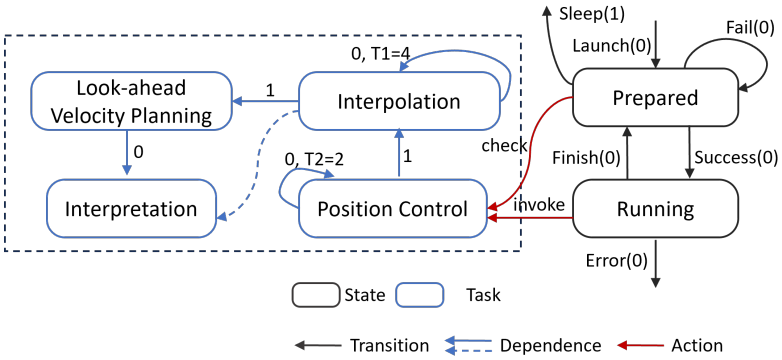



**Figure 10.** Graphical representation of the proposed open CNC system's control domain.

In this open CNC demonstration, the Interpretation task computes the tool trajectory parameters, such as each block's length (or radian), the angle between consecutive blocks, etc., in addition to executing the functions mentioned in Section 3.1. The Look-ahead Velocity-Planning task computes the feed limitation of the tool trajectory under given kinematic constraints (e.g., machine tool's maximum feed/acceleration/jerk, chord error, etc.). It is noticed that, if the kinematic constraints change (e.g., feed override), the Look-ahead Velocity-Planning task will be re-called to compute the feed limitation of the remaining tool trajectory under new kinematic constraints. The Interpolation task computes the velocity profile along the tool trajectory. Then, it computes the moving displacement in $T_1$ and obtains the next position along the tool trajectory. Moreover, the functionality of the Position Control task is the same as the modern CNCs mentioned in Section 3.1. In the Running state, therefore, the invocation action's sequence is Interpretation -> Look-ahead Velocity Planning -> Interpolation -> Position Control -> Interpolation -> Position Control -> ... It can be seen that a part of the Interpolation task and of the Position Control task are independent of the tool trajectory's shape.

According to the above control domain's formal description, the Interpolation task and the Position Control task have high time sensitivity (4 ms and 2 ms, respectively), while the other two tasks do not. The Look-ahead Velocity-Planning task is special since it needs to generate the new feed limitation for the Interpolation task in the case of kinematic constraints' change. That means the dependence between the Look-ahead Velocity-Planning task and the Interpolation task is intimate. Therefore, the control task virtualization is designed as follows. The Interpretation task is virtualized by the docker (the most popular container engine) and deployed on a cloud platform. The dependence on it is implemented as an HTTP/HTTPS request over the network. The Look-ahead Velocity-Planning task is also containerized by the docker, but deployed on a factory-level edge platform. The dependence on it is implemented as a TCP request over a local area network. Moreover, the Interpolation task is virtualized by wasm and deployed on an actuator-level edge platform. In detail, the velocity profile, displacement, and next position computation parts of the Interpolation task are compiled into a wasm module since they are compute-intensive algorithms that do not need system calls. The other part of the Interpolation task (e.g., interactions with sensors via the I/O) is developed as the host application. Finally, the Position

Control task migrates to each feed axis's servo driver. In other words, the position control mode of each servo driver is on. The dependence between the Interpolation task and the Position Control task is implemented via a kind of real-time communication, like EtherCAT.

Since the host application of the Interpolation task and the Position Control task are independent of the tool trajectory's shape, they can remain unchanged when the open CNC demonstration is modified to process the B-spline tool trajectory. In other words, only the Interpretation container, the Look-ahead Velocity-Planning container, and the Interpolation wasm module need to be replaced. In order to schedule these tasks, an additional scheduler task is developed in this open CNC demonstration. The activities of the scheduler task are illustrated via the activity diagram presented in Figure 11. Firstly, the scheduler task reads the configuration file to construct the open CNC system and publishes the initial state. Then, it listens to events, and meanwhile, it invokes the actions of the initial state. Regarding the listening of events, if the received event is not a registered event, the scheduler task will generate an error message; otherwise, it publishes the next state and invokes corresponding actions. Regarding the invocation of actions, the topological sorting of dependences between tasks is implemented first before the sequential orchestration of tasks according to the action's name. Take the check action associated with the Prepared state as an example: the scheduler task checks the availability (or accessibility) of the Interpretation task, Look-ahead Velocity-Planning task, Interpolation task, and Position Control task in the dependent sequence, according to their addresses in the configuration file. Therefore, change the addresses of the Interpretation container, the Look-ahead Velocity Planning container, and the Interpolation wasm module in the open CNC demonstration's configuration file, and the functionality can be modified to process different tool trajectories.

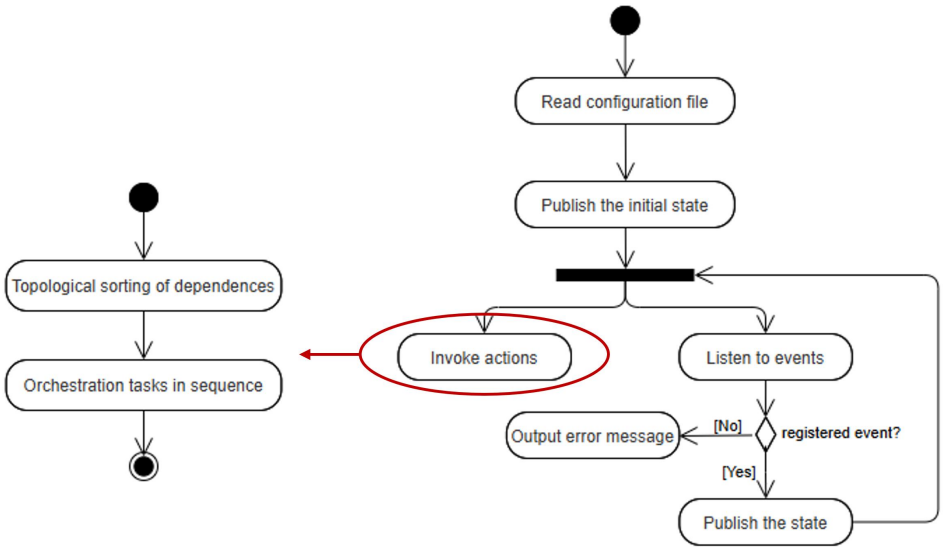



**Figure 11.** Activity diagram of the scheduler task.

### *4.2. Discussion*

From the proposed open CNC demonstration, it can be seen that, compared to current open control prototypes, the proposed open control system implementation method has two significant characteristics:

- It does not aim to construct an open control system according to a fixed control task granularity standard. Instead, it is based on the control domain's formal description, which is written in a configuration file. In the sense of the openness criteria, the proposed implementation method improves interoperability and extendibility by elevating the abstract level.
- It concentrates on adopting virtualization technologies to decouple control tasks from heterogeneous infrastructures as much as possible. The addresses of the virtualized control tasks are written in a configuration file. Therefore, compared to component-

and service-based open control prototypes, the proposed implementation method supports higher portability and scalability.

The proposed open control system implementation method, moreover, enables the break of the traditional control system's stand-alone operation pattern. For instance, the Interpretation container on the cloud in the above open CNC demonstration can be accessed by multiple Look-ahead Velocity-Planning containers from different factory-level edges. Further, the Look-ahead Velocity-Planning container can serve multiple Interpolation wasm modules from different actuator-level edges.

## 5. Conclusions and Future Work

In order to break the closed architecture of modern control systems in industrial automation, this paper presents a new implementation method to construct open control systems in the environment of the industrial IoT. Current open control systems are reviewed according to the open system definition and the criteria of openness, and two key issues are summarized: the decoupling of control tasks between each other and the decoupling of control tasks from heterogeneous infrastructures. Regarding the first issue, in contrast to current open control prototypes that depend on defining a control task granularity standard, this paper suggests relying on the control domain's formal description to decouple control tasks. Given that modern control systems often have both discrete and continuous behaviors, a hierarchical semantic model based on an extended FSM and a dependency network model with the time property is proposed. Regarding the second issue, moreover, unlike current open control prototypes that mostly develop control tasks in the form of components or services to achieve openness, this paper decouples control tasks from infrastructures via virtualization technologies. By discussing the merits and limitations of different virtualization technologies, and given the heterogeneity of infrastructures and the complexity of control domain, this paper suggests a hybrid virtualization approach based on containers and wasm. An open CNC demonstration is constructed by the proposed implementation method and compared to current open control prototypes from the openness criteria point of view.

In summary, from the control system perspective, the proposed implementation method allows the dynamic customization of control systems according to the production requirements, which contributes to responding to the mass customization challenge. From the infrastructure resource perspective, the proposed implementation method enables the improvement of the resource utilization rate, which contributes to green manufacturing.

In the future, the following works will be studied in sequence:

- The verification of the proposed hierarchical semantic model and the development of the control domain formal description application with a user interface.
- The study of unikernel-based control task virtualization to complement containers and wasm, and the theoretical analysis of the hybrid virtualization approach to maximize the utilization of infrastructure resources in the industrial IoT.
- A corresponding operation pattern will be designed to replace the traditional stand-alone operation pattern, and this new operation pattern will integrate the implementation of the scheduler task.

**Author Contributions:** Conceptualization, L.L. and X.Q.; methodology, L.L.; investigation, L.L. and Z.X.; writing—original draft preparation, L.L.; writing—review and editing, X.Q.; funding acquisition, L.L. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by the Science, Technology and Innovation Commission of Shenzhen Municipality No. 20220809175919001.

**Data Availability Statement:** Data are contained within the article.

**Conflicts of Interest:** Lisi Liu has not received research grants from China Nuclear Power Technology Research Institute Co., Ltd. However, Xiaobin Qu from China Nuclear Power Technology Research Institute Co., Ltd. has been involved in this study as a consultant.

## Appendix A

Part of the configuration file of the proposed open CNC system:

```
{
  "name": "openCNC",
  "initialState": "idle"
}
{
  "name": "prepared",
  "transitions": {
    "toIdle": {
      "trigger": {
        "name": "sleep"
      },
      "priority": 1,
      "sourceState": "prepared",
      "targetState": "idle"
    },
    "toPrepared": {
      "trigger": {
        "name": "fail"
      },
      "priority": 0,
      "sourceState": "prepared",
      "targetState": "prepared"
    },
    "toRunning": {
      "trigger": {
        "name": "success"
      },
      "priority": 0,
      "sourceState": "prepared",
      "targetState": "running"
    }
  },
  "actions": {
    "check": {
      "startTask": {
        "name": "position control",
        "dependences": {
          "pcself": {
            "time": 1.0,
            "priority": 0,
            "sourceTask": "position control",
            "targetTask": "position control"
          },
          "pcOnInterpo": {
            "time": 0.0,
            "priority": 0,
            "sourceTask": "position control",
            "targetTask": "interpolation"
          }
        }
      }
```

```
        }
    }
}
```

## References


1. Pritschow, G.; Altintas, Y.; Jovane, F.; Koren, Y.; Mitsuishi, M.; Takata, S.; Van Brussel, H.; Weck, M.; Yamazaki, K. Open controller architecture—Past, present and future. *CIRP Ann.* **2001**, *50*, 463–470. [CrossRef]
2. Michaloski, J.; Birla, S.; Yen, C.J.; Igou, R.; Weinert, G. An open system framework for component-based CNC machines. *ACM Comput. Surv.* **2000**, *32*, 23. [CrossRef]
3. Park, S.; Kim, S.-H.; Cho, H. Kernel software for efficiently building, re-configuring, and distributing an open CNC controller. *Int. J. Adv. Manuf. Technol.* **2005**, *27*, 788–796. [CrossRef]
4. Ma, X.; Han, Z.; Wang, Y.; Fu, H. Development of a PC-based Open Architecture Software-CNC System. *Chin. J. Aeronaut.* **2007**, *20*, 272–281. [CrossRef]
5. Liu, L.; Yao, Y.; Li, J. A review of the application of component-based software development in open CNC systems. *Int. J. Adv. Manuf. Technol.* **2020**, *107*, 3727–3753. [CrossRef]
6. Givehchi, O.; Imtiaz, J.; Trsek, H.; Jasperneite, J. Control-as-a-service from the cloud: A case study for using virtualized PLCs. In Proceedings of the 10th IEEE Workshop on Factory Communication Systems, Toulouse, France, 5–7 May 2014.
7. Bigheti, J.A.; Fernandes, M.M.; Godoy, E.P. Control as a Service: A microservice approach to industry 4.0. In Proceedings of the II Workshop on Metrology for Industri 4.0 and IoT, Naples, Italy, 4–6 June 2019.
8. *IEC 61499*; The New Standard in Automation. International Electrotechnical Commission: Geneva, Switzerland, 2024. Available online: https://iec61499.com/ (accessed on 24 July 2024).
9. Minhat, M.; Vyatkin, V.; Xu, X.; Wong, S.; Al-Bayaa, Z. A novel open CNC architecture based on STEP-NC data model and IEC 61499 function blocks. *Robot. Comput.-Integr. Manuf.* **2009**, *25*, 560–569. [CrossRef]
10. Huang, X. Enhancing STEP-NC compliant CNC controller for distributed and reconfigurable environment in production line. In Proceedings of the 2010 International Conference on Computer, Mechatronics, Control and Electronic Engineering, Changchun, China, 24–26 August 2010.
11. Vyatkin, V. The IEC 61499 standard and its semantics. *IEEE Ind. Electron. Mag.* **2009**, *3*, 40–48. [CrossRef]
12. Dai, W.; Pang, C.; Vyatkin, V.; Christensen, J.M.; Guan, X. Discrete-event-based deterministic execution semantics with timestamps for industrial cyber-physical systems. *IEEE Trans. Syst. Man Cybern.* **2020**, *50*, 851–862. [CrossRef]
13. Prenzel, L.; Zoitl, A.; Provost, J. IEC 61499 runtime environments: A state of the art comparison. *Comput. Aided Syst. Theory* **2020**, *12014*, 453–460.
14. Dai, W.; Zhang, Y.; Kong, L.; Christensen, J.H.; Huang, D. Design of industrial edge applications based on IEC 61499 microservices and containers. *IEEE Trans. Ind. Inform.* **2022**, *19*, 7925–7935. [CrossRef]
15. Cengic, G.; Akesson, K. On formal analysis of IEC 61499 applications, part A: Modeling. *IEEE Trans. Ind. Inform.* **2010**, *6*, 136–144. [CrossRef]
16. Cengic, G.; Akesson, K. On formal analysis of IEC 61499 applications, part B: Execution semantics. *IEEE Trans. Ind. Inform.* **2010**, *6*, 145–154. [CrossRef]
17. Scordino, C.; Savino, I.M.; Cuomo, L.; Miccio, L.; Tagliavini, A.; Bertogna, M.; Solieri, M. Real-time virtualization for industrial automation. In Proceedings of the 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vienna, Austria, 8–11 September 2020.
18. Xi, S.; Wilson, J.; Lu, C.; Gill, C. RT-Xen: Towards real-time hypervisor scheduling in Xen. In Proceedings of the Ninth ACM International Conference on Embedded Software (EMSOFT), Taipei, China, 9–14 October 2011.
19. Xi, S.; Xu, M.; Lu, C.; Phan, L.T.; Gill, C.; Sokolsky, O.; Lee, I. Real-time multi-core virtual machine scheduling in Xen. In Proceedings of the International Conference on Embedded Software (EMSOFT), New Delhi , India, 12–17 October 2014.
20. Queiroz, R.; Cruz, T.; Mendes, J.; Sousa, P.; Simões, P. Container-based virtualization for real-time industrial systems—A Systematic Review. *ACM Comput. Surv.* **2023**, *56*, 1–38. [CrossRef]
21. Struhár, V.; Behnam, M.; Ashjaei, M.; Papadopoulos, A.V. Real-time containers: A survey. *Open Access Ser. Inform.* **2020**, *80*, 7:1–7:9.
22. Kivity, A.; Laor, D.; Costa, G.; Enberg, P.; Har'El, N.; Marti, D.; Zolotarov, V. OSv: Optimizing the operating system for virtual machines. In Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference (USENIX ATC'14), Berkeley , CA, USA, 19–20 June 2014.
23. Kuenzer, S.; Bădoiu, V.A.; Lefeuvre, H.; Santhanam, S.; Jung, A.; Gain, G.; Soldani, C.; Lupu, C.; Teodorescu, Ş; Răducanu, C.; et al. Unikraft: Fast, specialized unikernels the easy Way. In Proceedings of the 16th European Conference on Computer Systems, Virtual , 26–28 April 2021.
24. Haas, A.; Rossberg, A.; Schuff, D.L.; Titzer, B.L.; Holman, M.; Gohman, D.; Wagner, L.; Zakai, A.; Bastien, J.F. Bringing the web up to speed with WebAssembly. In Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, Barcelona, Spain, 18–23 June 2017.
25. Watt, C.; Rao, X.; Pichon-Pharabod, J.; Bodin, M.; Gardner, P. Two mechanisations of WebAssembly 1.0. In *Formal Methods*; Springer EBooks; Springer: Cham, Switzerland , 2021; Volume 13047, pp. 61–79.

26. Ray, P.P. An overview of WebAssembly for IoT: Background, tools, state-of-the-art, challenges, and future directions. *Future Internet* **2023**, *15*, 275. [CrossRef]
27. Wallentowitz, S.; Kersting, B.; Dumitriu, D.M. Potential of WebAssembly for embedded Systems. In Proceedings of the 11th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, 7–10 June 2022.

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.