Multi-agent Based IEC 61499 Function Block Modelling for Distributed Intelligent Automation

Guolin Lyu(✉) and Robert W. Brennan

University of Calgary, 2500 University Drive NW, Calgary T2N 1N4, Canada
guolin.lyu@ucalgary.ca

Abstract. A major challenge for traditional systems is the lack of capabilities to automatically discover alternative solutions and actively deploy corresponding functions to intelligently adapt to changes in dynamic environments. In this paper, we continue our previous research of developing a two-layer architecture for modelling industrial cyber-physical systems and focus on the IEC 61499 function block based low-level physical module design. We propose an architecture model to integrate function blocks with intelligent agents to support self-management capabilities for low-level physical modules to quickly respond to changes. In the proposed architecture, a self-manageable service model is introduced for IEC 61499 function block modelled systems and designed as a multi-agent model with *Self-Manageable Service Execution Agent*, *Self-Configuration Agent*, *Self-Healing Agent*, *Self-Optimization Agent*, and *Self-Protection Agent*. The proposed modelling framework is tested with preliminary experiments on Raspberry Pi using the agent modelling tool SPADE and function block modelling tool Eclipse 4diac.

Keywords: Distributed automation · Function block · IEC 61499 · Multi-agent modelling · Self-Manageability

1 Introduction

A key challenge with the transition to Industry 4.0 [1], is that of designing industrial automation systems that are responsive and adaptive to changes. This is particularly problematic in situations where low-level physical modules must interact in a timely manner with high-level cyber modules and manage the exchange of high-volume streaming data. With recent advances in system hardware and software, and enabling technologies for system computation and communication, traditional systems are being transformed into a new form: i.e., industrial cyber-physical systems (iCPS). iCPS are envisioned as a collaboration of cyber, physical components and their surrounding environments, that are empowered for intelligence by computing and communicating cores [2]. This new type of iCPS appears to hold the most promise of achieving modern industrial automation systems in the Industry 4.0 era, to be flexible in distribution of architectures, and to be intelligent in adaptation to changes in dynamic environments.

K.-Y. Kim et al. (Eds.): FAIM 2022, LNME, pp. 395–407, 2023.
https://doi.org/10.1007/978-3-031-17629-6_41

Central to this work is research on integration and interaction of high-level intelligent software agents and low-level real-time control units for iCPS [3]. In our previous research, we proposed a two-layer architecture design for modelling self-manageable iCPS for distributed intelligent automation [4]. A multi-agent computing model was designed to implement the high-level architecture with the aim of providing system intelligence by communicating and computing cores for cyber modules. The IEC 61499 function block (FB) model was used to implement the low-level architecture and offer real-time adaptation by distributed and intelligent control of physical modules [4]. In [4], only details of the high-level cyber module were explored.

In this paper we focus on the IEC 61499 FB based low-level architecture modelling and propose a design model to integrate function blocks with intelligent agents to support self-management capabilities of the low-level physical module to realize real-time adaptation to changes. In the proposed low-level architecture model, a self-manageable service model is introduced for IEC 61499 FB modelled systems and designed as a multi-agent model. The proposed modelling framework is tested with preliminary experiments on Raspberry Pi by using agent modelling tool SPADE and IEC 61499 FB modelling tool Eclipse 4diac.

2 Related Work

Agent-based solutions or multi-agent modelling techniques are playing a key role in the development of complex industrial control and automation systems, allowing a decentralized way to design distributed and intelligent systems [5, 6]. This approach is being applied in several domains of industrial applications: e.g., factory and building automation, power and energy systems [5]. Multi-agent modelling applies multiple intelligent, autonomous, and cooperative agents in system design as these agents have capabilities of acting independently or in collaboration with each other to respond to system requests or changes and to achieve individual or shared goals [6].

Real-time control units or automation functions are mainly programmed under IEC 61131-3 [7] or IEC 61499 [8] and executed in the programmable logic controllers or intelligent embedded devices. IEC 61131-3 dominates the design of traditional industrial automation systems and faces many challenges, while IEC 61499 is developed for programming next-generation industrial automation systems to support portability, interoperability, and configurability [8, 9]. Compared to IEC 61131-3, IEC 61499 offers key features including application-based distributed architecture design, object-oriented function block modelling, and event-driven control application execution.

In this section, we will focus on reviewing studies on applying multi-agent modelling techniques to develop IEC 61499 FB models for self-manageable iCPS: especially aiming at the low-level control architecture modelling. Recent key research includes the European Daedalus project for developing IEC 61499 based distributed control and simulation platforms for iCPS [10], the IEEE standard of interface practice patterns for integration of software agents and low-level automation functions [3].

One major research trend in this area is the focus on automatic reconfiguration modelling of distributed automation systems. Brennan *et al.* [11] proposed a three-layered FB and agent-based model for dynamic and intelligent reconfiguration of real-time distributed control systems. A reconfigurable concurrent FB model was also proposed to

separate two control paths: IEC 61499 FB modeled control application execution and multi-agent modeled configuration control operation [12]. Khalgui *et al.* [13] proposed an architecture of reconfigurable multi-agent systems for IEC 61499 based distributed control systems. Two types of agents are provided: reconfiguration agents modeled by nested state machines for local automatic reconfiguration and coordination agents defined by coordination matrices and communication protocols for managing reconfiguration behaviors. Guellouz *et al.* [14] proposed a new design pattern reconfiguration FB (RFB) which is defined as event-triggered software components to control and execute reconfiguration tasks. Compared to traditional IEC 61499 FB model, it adds reconfiguration events to the interface and the master-slave execution control chart to define reconfiguration functions.

Recently, more attention has been focused on system autonomic service management modelling. An agent-based architecture for self-manageable industrial automation systems was proposed by Mubarak and Göhner [15], in which agents are deployed on three levels including the control and supervision level, the self-management functionality level, and the automation system connection level. An automation agent and IEC 61499 FB based architecture for low-level physical modules control was proposed by Lepuschitz *et al.* [16], aiming at realizing system self-reconfiguration. A self-represented architecture to support system self-reconfiguration and autonomous monitoring was proposed by Kaindl *et al.* [17], in which the lower-level control is modeled by IEC 61499 FBs and the high-level control is designed as multi-agents. A self-organizing architecture was proposed by Strasser and Froschauer [18] to support automatic reconfiguration of new devices and automatic deployment of control applications for IEC 61499 based automation systems. Dai *et al.* [19] proposed a knowledge-driven autonomic service management architecture modelling method by using service-oriented self-manageable agents on the device level for continuously optimizing physical resource utilization. Dai *et al.* [20] proposed a cloud-based decision support system design approach by applying autonomous industrial software agents for system real-time failure detection and online reconfiguration. The analysis technique fault trees and control algorithms are modeled in IEC 61499 FBs for execution in both physical controllers and cloud services. The above reviewed studies primarily focused on service-oriented self-manageable iCPS architecture modelling and some self-management capabilities, e.g., self-configuration [15, 18], self-optimization [19], self-healing [20], were demonstrated through case studies.

In summary, industrial agent-based solutions are not uncommon to be applied in the high-level distributed automation systems as it has the capabilities that are well matched to the high-level core tasks, e.g., strategic decision-making on operation, planning, scheduling, and maintenance. However, very little work has been done on integration of multi-agent modelling with low-level control design due to stringent requirements for system reliability, timeliness, safety, etc. Some other challenges come from, for example, traditional system architecture modelling and control application design approaches limit such integration. Recently, IEC 61499 has been integrated with its enabling technologies for design and computing (e.g., service-oriented architecture [21], autonomic computing [22]) to realize distributed and intelligent automation for iCPS. Inspired by

the above research, we aim at proposing a multi-agent based IEC 61499 FB architecture by integrating function blocks with intelligent agents to support self-management capabilities for low-level physical modules to realize real-time adaptation to changes.

3 Proposed Framework

In our previous research, a two-layer architecture for modelling iCPS was developed, in which the high-level cyber module was designed and implemented as a multi-agent computing model of *Monitoring Agent*, *Analysis Agent*, *Planning Agent*, *Execution Agent*, *Self-Learning Agent*, and *Knowledge Agent* (i.e., multi-agent MAPLE-K model) [4]. For the low-level physical module implementation, we propose utilizing the IEC 61499 FB model with embedded intelligent agents for real-time distributed control. The agent-FB interface design can be referred to IEEE standard 2660.1 [3] and its related work (e.g., [23]). In this section, we begin with a brief overview of the IEC 61499 reference architecture, then describe our proposed modelling framework.

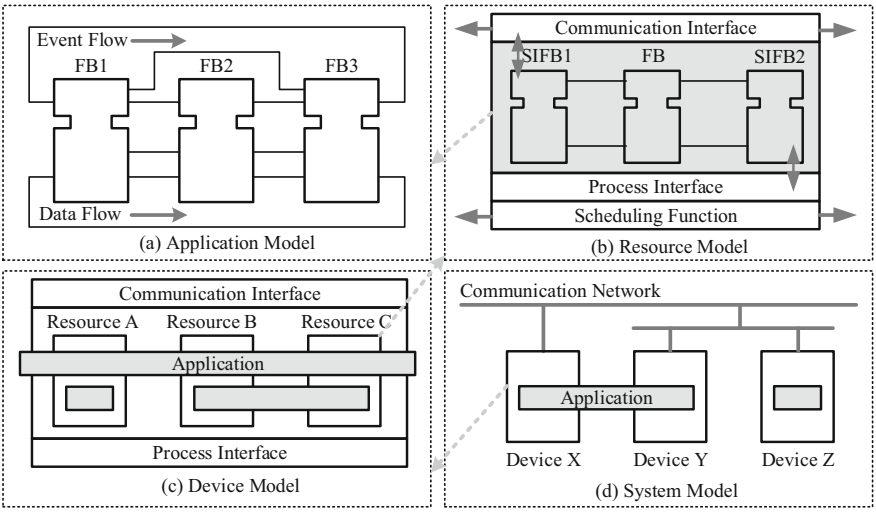

Fig. 1. The IEC 61499 reference architecture

3.1 IEC 61499 Function Block Model

Figure 1 provides an overview of the IEC 61499 reference architecture [8]. Generally, a typical IEC 61499 based system is designed as: a) the control logic built by *function blocks* as *applications*, and b) the physical *devices* encapsulating required *resources* for implementation. A *function block* is an object-oriented modelling element with event-driven execution. An *application* model (Fig. 1a) is defined as a network of interconnected FBs linked by event/data flows and distributed over resources and devices. A

resource model (Fig. 1b) is defined to support the execution of one or more application fragments. A *device* model (Fig. 1c) is defined to support one or more resources to exchange data through interface services internally (i.e., the process interface to enable interaction via input/output points in local devices) and externally (i.e., the communication interface to enable interaction via networks with resources in remote devices). A *system* model (Fig. 1d) is a collection of interconnected devices interacting with each other through communication networks.

3.2 Self-manageable Service Model

In the proposed low-level architecture modelling framework (Fig. 2), the self-manageable service model is design as a multi-agent model embedded in IEC 61499 FBs. Two new agent types are designed (Table 1): a) the self-manageable service execution agent *Agent_SMS*; b) self-manageable agents including *Agent_SC*, *Agent_SO*, *Agent_SH*, and *Agent_SP*. In general, the first type of agent is mainly responsible for communicating with the multi-agent MAPLE-K model in the cyber module on current states and change requests, and with the self-manageable agents on self-manageable service action plans, and is also responsible for finally executing the action plan on IEC 61499 FB based systems. The second type of agent is primary concerned with generating self-manageable service action plans upon requests. For example (Fig. 2), the basic process of one type of self-configuration is instigated by a single change request that results in the old *FB_S* being replaced by two new *FB_S1* and *FB_S2* in the *Application* residing in *Device_Y* and *Device_Z*. The self-manageable service is activated as *Agent_SMS* detects the change request and communicates to *Agent_SC* to request the self-configuration service. *Agent_SC* generates the action plan and sends it back to *Agent_SMS* for execution on the application.

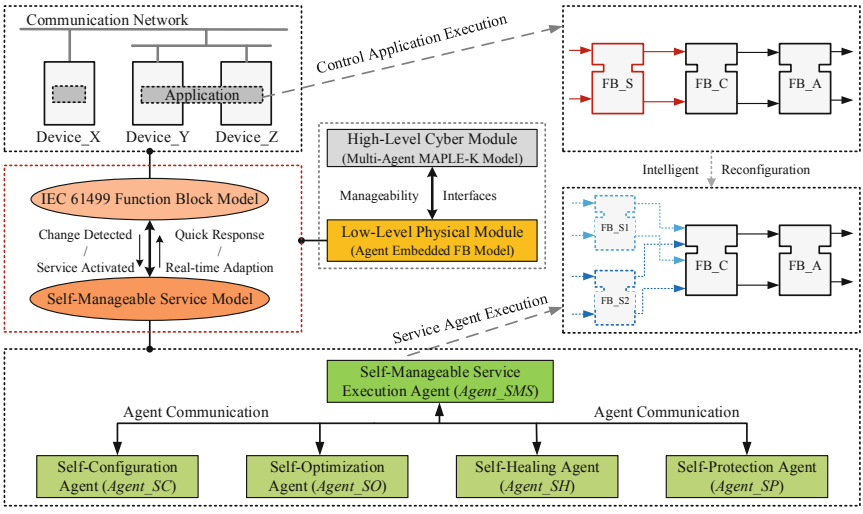

Fig. 2. The proposed low-level architecture modelling framework

Table 1. An overview of proposed self-manageable agents

Name	Description
Self-Manageable Service Execution Agent (*Agent_SMS*)	Monitoring system states and responding to changes by deciding the adequate behaviors to perform (i.e., activate one or more self-manageable agents and execute self-manageable services)
Self-Configuration Agent (*Agent_SC*)	Configuring/reconfiguring functions, structures, and process to adapt to dynamical changes
Self-Optimization Agent (*Agent_SO*)	Improving and optimizing performances and operations with respect to predefined goals
Self-Healing Agent (*Agent_SH*)	Detecting and recovering from disturbances and faults to maximize system availability
Self-Protection Agent (*Agent_SP*)	Identifying and protecting against safety and security attacks to preserve system integrity

***SelfManageableServiceExecutionAgent* Class**. *Agent_SMS* plays a key role in requesting one or more self-manageable agents to respond to changes and executing self-manageable services provided by self-manageable agents to adapt IEC 61499 FB based systems. In the UML data model in Fig. 3, the *Agent_SMS* class implements two interfaces (i.e., the *SelfManageableAgents* interface and the *IEC61499FunctionBlockSystem* interface) to realize its predefined functions. Typical attributes of the *Agent_SMS* class include *previousState*, *executedAction*, *currentState*, *plannedAction*, and *computedAction*. Typical methods are *receiveCurrentState*, *initializeSMAgent*, and *executeAgentSMS*.

***SelfManageableAgents* Interface**. The *SelfManageableAgents* interface in Fig. 4 provides access to communicate with self-manageable agents for self-manageable services in the low-level physical module. One typical attribute is the list of self-manageable agent types (i.e., *Agent_SC*, *Agent_SO*, *Agent_SH*, and *Agent_SP*). Each agent is responsible for their own tasks as shown in Table 1 [22]. Typical methods are *receiveChangeRequest* to get the change request from *Agent_SMS*, *selectSMAgents* to choose which agent to perform the self-manageable service according to *currentState* and *changeRequest*, and *sendReqDecision* to ask the chosen agent to perform required tasks. For each self-manageable agent class, typical methods are to receive the request decision from the *SelfManageableAgents* interface, generate the self-manageable service action plan, and return it to *Agent_SMS* for execution.

An example of sensor nodes in wireless sensor networks [4] is discussed from a point of view of the low-level physical module to explain how the *SelfManageableAgents* interface works for the *SelfManageableServiceExecutionAgent* class. In normal operations, as per the system's request, *Agent_SO* can be initiated to provide an optimized service plan for *Agent_SMS* (e.g., sensor nodes request high resource consumption only when changes detected, or become passive to save batteries when no changes happen in a

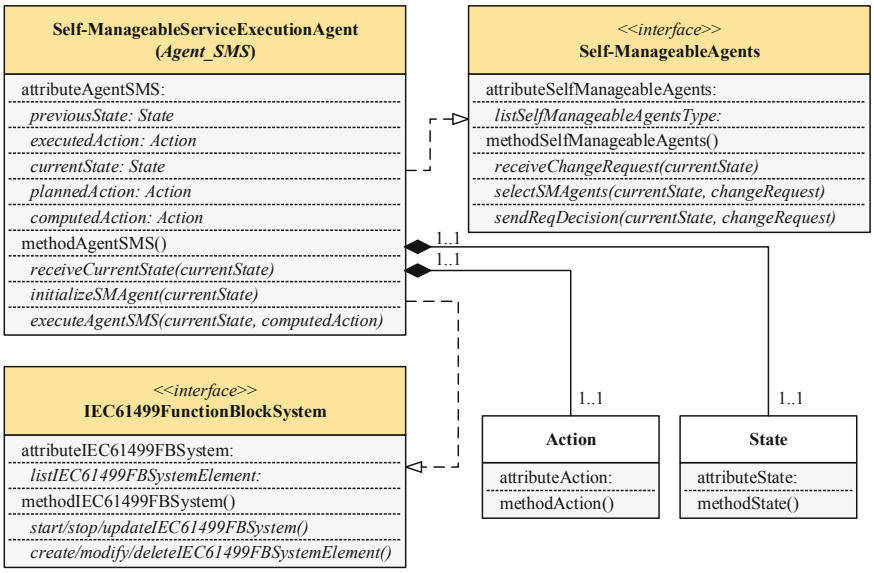

Fig. 3. The *SelfManageableServiceExecutionAgent* class

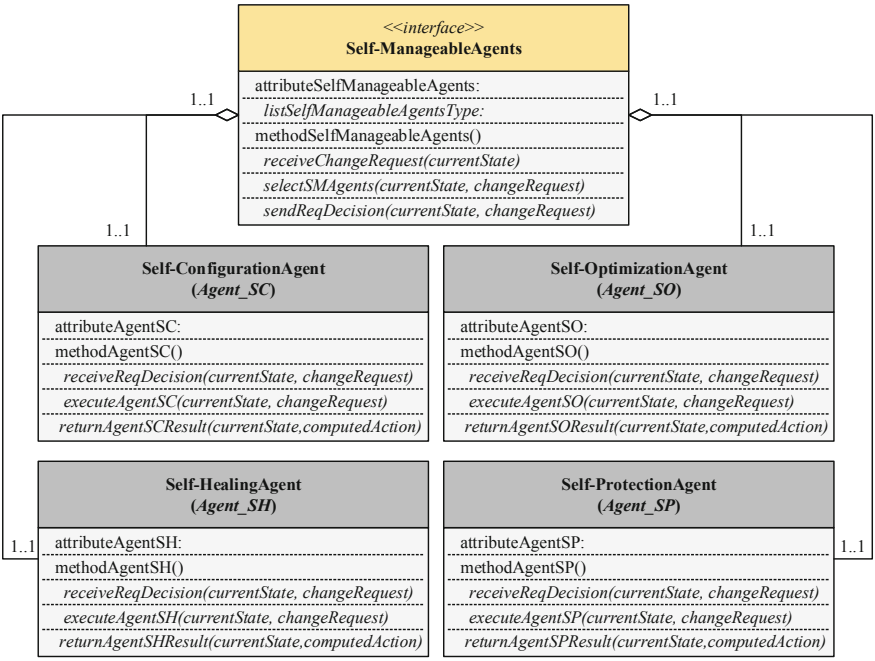

Fig. 4. The *SelfManageableAgents* interface

period of time). For more complex situations (e.g., change of power sources in sensor nodes), as requested from the system, *Agent_SMS* can initiate *Agent_SH* or *Agent_SP*. *Agent_SP* gives out warning signals in an unrecoverable situation where both the primary and backup batteries are running out, and *Agent_SH* activates the backup battery plan in a recoverable situation where only the primary batter is running out. If more sensor nodes are required to be added to the network, *Agent_SC* will be activated. *Agent_SC* can also be initialized by *Agent_SMS* in execution of service plans from *Agent_SO*, *Agent_SH*, and *Agent_SP*.

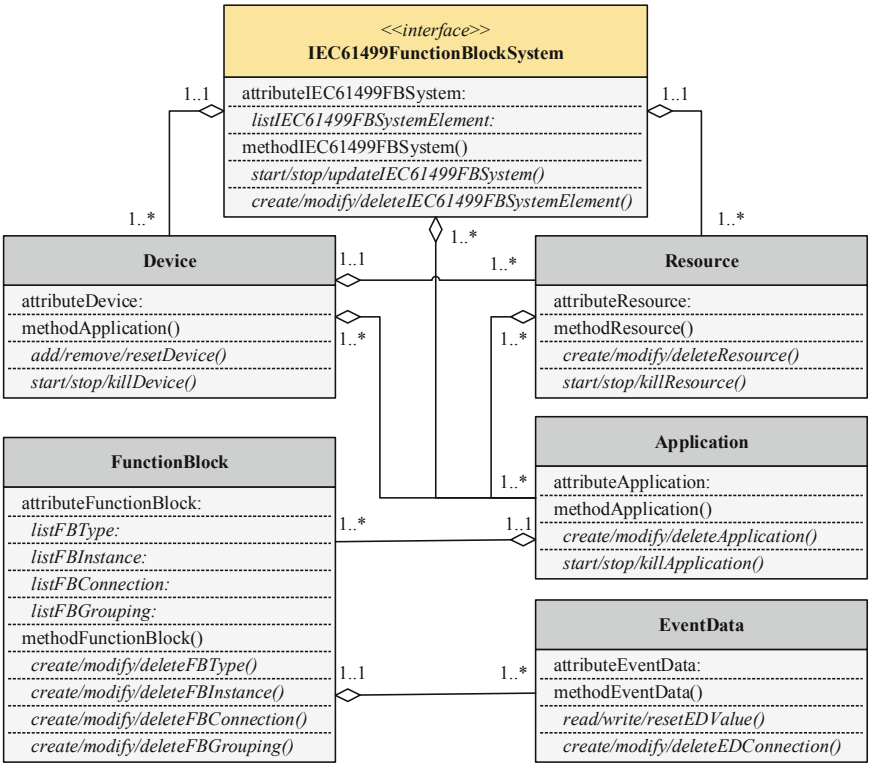

Fig. 5. The *IEC61499FunctionBlockSystem* interface

***IEC61499FunctionBlockSystem* Interface**. The *IEC61499FunctionBlockSystem* interface in Fig. 5 provides access to executing self-manageable services to adapt IEC 61499 FB based systems. The typical attribute is a list of IEC 61499 FB system elements, e.g., devices, resources, applications, and function blocks. Typical methods implemented are *start/stop/updateIEC61499FBSystem* and *create/modify/deleteIEC61499FBSystemElement*. The classes *Device*, *Resource*, *Application*, *FunctionBlock* with *Event/Data* are main entities that *Agent_SMS* can execute the self-manageable service action plan on. In the *Device* class, typical methods include *add/remove/resetDevice* and *start/stop/killDevice*. In the *Resource* class,

typical methods include *create/modify/deleteResource* and *start/stop/killResource*. For the *Application* class, typical methods are *create/modify/deleteApplication* and *start/stop/killApplication*. For the *EventData* class, typical methods are *read/write/resetEDValue* and *create/modify/deleteEDConnection*. For the *FucntionBlock* class, typical attributes are *FBType* (e.g., basic/composite/service interface FBs), *FBInstance*, *FBConnection*, and *FBGrouping* (e.g., plug and socket adapters), and typical methods are to create, modify, and delete those attributes.

Figure 6 provides an example of how the *IEC61499FunctionBlockSystem* interface works for the *SelfManageableServiceExecutionAgent* class: the calculator model with only two functions (i.e., *ADD* and *SUB*) needs to be reconfigured to add two more functions (i.e., *MUL* and *DIV*). In Fig. 6a, the calculator is built using a composite FB model: *Agent_SMS* works primarily on the *FunctionBlock* class to create two new FB instances and their connections, and updates attributes and methods in the *EventData* class for *FB1* (red dashed circles in Fig. 6a). For the calculator designed as a basic FB (Fig. 6b), the self-configuration process is easier: main tasks of *Agent_SMS* involve adding two events and updating execution control charts and algorithms in the *EventData* class for *Calculator 1.0* (red dashed circles in Fig. 6b). Thus, for change requests in different designs and systems, tasks of *Agent_SMS* will be different: e.g., to add event/data, to create FB connections, to modify applications, to rebalance resources, and to reset devices.

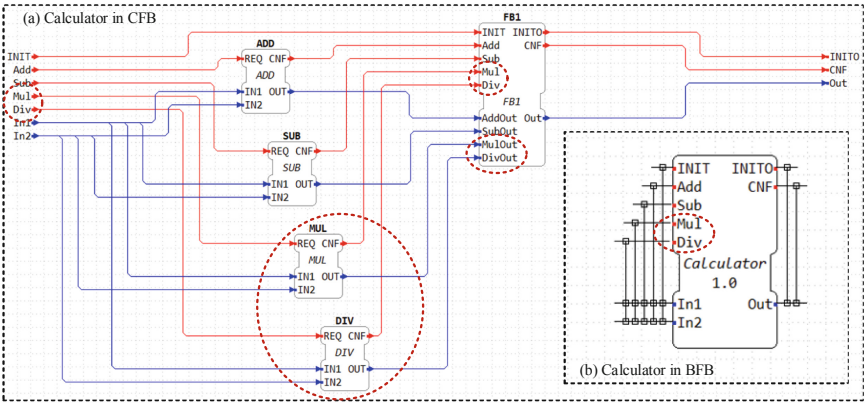

Fig. 6. An example of the calculator model built in Eclipse 4diac

3.3 Agent_Embedded Function Block Model

The agent-embedded function block (Fig. 7) is proposed as a new design pattern for IEC 61499 to build self-manageable automation and control solutions. The basic FB type is used to create a new type called *Agent_X FB* including *Agent_SMS FB*, *Agent_SC FB*, *Agent_SO FB*, *Agent_SH FB*, and *Agent_SP FB*. According to data models in Figs. 3, 4 and 5, *Agent_SMS FB* has at least two key state/action pairs in its execution control chart: a) REQ for requesting one or more self-manageable agents to respond to changes (implementing the *SelfManageableAgents* interface); b) EXE for executing self-manageable

services provided by self-manageable agents to adapt FB based control systems (implementing the *IEC61499FunctionBlockSystem* interface). Self-manageable agents embedded in IEC 61499 FBs are either initialized to be active for expected tasks or deactivated in a sleep state. With this new design pattern, the new agent-embedded FB types can be introduced to build self-manageable control applications: e.g., replacing the failed software component automatically. In practice, in order for one IEC 61499 FB based application to be self-manageable, the *Agent_SMS FB* type is required with one or more self-manageable agents for different tasks (i.e., configuration, optimization, healing, or protection). The previous sub-sections have explained how it works, especially the examples of the calculator and sensors nodes modelling.

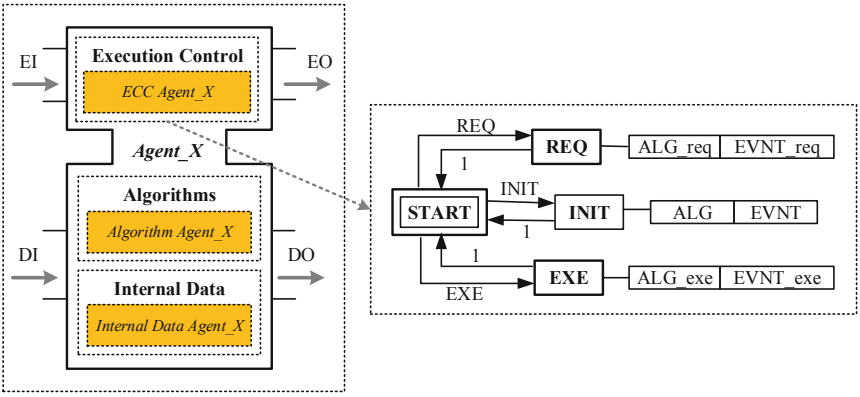

Fig. 7. The agent-embedded function block model

4 Proposed Experiment

In this section, a preliminary test setup and different scenarios are provided to illustrate the proposed modeling framework and some of the self-management features are also demonstrated. The design tool SPADE (Smart Python Agent Development Environment) [24] is used to develop multi-agent models. The Eclipse 4diac design tool [25] is used to develop IEC 61499 FB models. The proposed test setup in Fig. 8 includes: a) the signaling platform represented by Raspberry Pi [26] (#3 in Fig. 8) powered LEDs (#2 in Fig. 8) in blue, green, red, and yellow; b) the sorting platform represented by Raspberry Pi (#3 in Fig. 8) powered Motors (#4 in Fig. 8); and c) the carrying platform represented by Jetson Nano powered JetBot [28] (#1 in Fig. 8). In this case, JetBot simply represents the high-level cyber module and is able to follow a path and communicate with other devices. LEDs and Motors powered by Raspberry Pi represents low-level physical modules and are controlled by agent-embedded IEC 61499 FB modeled applications.

For the test process, the signaling platform randomly selects a colored LED and turns it on for a few seconds. The color and duration are communicated to the carrying platform. Next, JetBot moves to the sorting platform and sends a message to the Motor,

specifying the rotation direction and duration. The Motor executes the command and sends back a confirmation message to JetBot.

To test the system self-manageable capabilities, we begin with a simple single direction of rotation scenario: only a blue LED randomly flashes to request clockwise rotation of Motor1. This is extended to multi-direction rotation: a green LED is added for the counterclockwise rotation of Motor1. Applying the proposed modeling framework, new IEC 61499 FBs can be added to self-configure control applications, and resources can be re-distributed accordingly to support this change.

A more complicated scenario is to add another sorting line (i.e., Motor2) for low-speed rotation with high precision. The existing control system can be easily reconfigured and redeployed to the new line with the proposed modeling framework. Self-optimization can also be achieved in both lines due to operation data collected from the old line. One self-healing case is the system can quickly update its IEC 61499 FB modeled application when detecting that the blue LED is broken and replaced with a yellow one. For self-protection features, for example, a much heavier box blocks the line operation, and the system will automatically stop for protection.

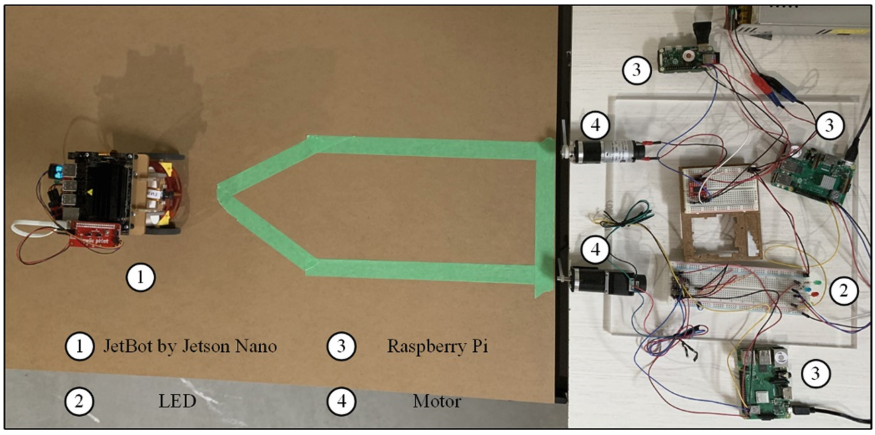

Fig. 8. The preliminary experiment setup

5 Conclusion and Future Work

In this paper we continued our previous research of developing a two-layer architecture for modelling iCPS and focused on the IEC 61499 FB based low-level physical module design. We proposed a conceptual design framework to model multi-agent embedded IEC 61499 FB based systems for distributed intelligent automation, in which intelligent agents are integrated with function blocks to support self-management capabilities of low-level physical modules to realize real-time adaptation to changes. In the proposed conceptual design, a self-manageable service model is introduced for IEC 61499 FB based systems and designed as a multi-agent model with *Agent_SMS*, *Agent_SC*,

Agent_SO, *Agent_SH*, and *Agent_SP*. A preliminary experiment setup was demonstrated for the proof-of-concept test and some scenarios for self-manageable features were also described. The proposed modelling framework can also help both cyber and physical modules to be more focused on their own core tasks (e.g., strategic decision support from the high-level cyber module for system performance optimization, and streaming data analysis in the low-level physical module for real-time adaptation to changes), thus, achieve balanced system capabilities by distributing system intelligence from the high-level cyber module to the low-level physical module. Continuing from this study, one key task is to develop detailed experiments to test and evaluate the proposed conceptual design, including programming agent-embedded IEC 61499 FB modelled applications and deploying them into intelligent devices for testing, and developing design metrics and running simulations for evaluation. Another key task is to find and choose best interface practices for communication between the high-level cyber module and low-level physical module.

Acknowledgement. The authors wish to thank the Natural Sciences and Engineering Research Council of Canada (NSERC), Spartan Controls, and the Suncor Energy Foundation for their generous support of this research through Chairs in Design Engineering (CDE) grant 486462–15.

References

1. Kagermann, H., Wahlster, W., Helbig, J.: Recommendations for implementing the strategic initiative INDUSTRIE 4.0: Securing the future of German manufacturing industry. In: Final Report of INDUSTRIE 4.0 Working Group, Frankfurt, Germany (Apr. 2013)
2. Cyber-Physical Systems. https://www.nsf.gov/funding/pgm_summ.jsp?pims_id=503286. Last accessed 20 Jan 2022
3. IEEE Recommended Practice for Industrial Agents: Integration of Software Agents and Low-Level Automation Functions, IEEE Standard, 2660.1–2020, Standards Committee of IEEE Industrial Electronics Society (Jan. 2021)
4. Lyu, G., Fazlirad, A., Brennan, R.W.: Multi-agent modeling of cyber-physical systems for IEC 61499 based distributed automation. Proc. Manuf. **51**, 1200–1206 (2020)
5. Leitao, P., Karnouskos, S., Ribeiro, L., Lee, J., Strasser, T., Colombo, A.W.: Smart agents in industrial cyber–physical systems. Proc. IEEE **104**(5), 1086–1101 (2016)
6. Mařík, V., McFarlane, D.: Industrial adoption of agent-based technologies. IEEE Intell. Syst. **20**(1), 27–35 (2005)
7. Programmable Controllers—Part 3: Programming Languages. International Standard, IEC 61131–3:2013, 3rd ed., International Electrotechnical Commission, Geneva, Switzerland (Feb. 2013)
8. Function Blocks—Part 1: Architecture, International Standard, IEC 61499–1:2012, 2nd ed., International Electrotechnical Commission, Geneva, Switzerland (Nov. 2012)
9. Vyatkin, V.: IEC 61499 as enabler of distributed and intelligent automation: State-of-the-art review. IEEE Trans. Industr. Inf. **7**(4), 768–781 (2011)
10. Daedalus. https://ecn.iec61499.eu/ecn-projects/view-full?id=12. Last accessed 20 Jan 2022
11. Brennan, R.W., Fletcher, M., Norrie, D.H.: An agent-based approach to reconfiguration of real-time distributed control systems. IEEE Trans. Robot. Autom. **18**(4), 444–451 (2002)
12. Brennan, R.W., Zhang, X., Xu, Y., Norrie, D.H.: A reconfigurable concurrent function block model and its implementation in real-time Java. Integr. Comput. Aided Eng. **9**(3), 263–279 (2002)

13. Khalgui, M., Mosbahi, O., Li, Z., Hanisch, H.M.: Reconfiguration of distributed embedded-control systems. IEEE/ASME Trans. Mechatron. **16**(4), 684–694 (2010)
14. Guellouz, S., Benzina, A., Khalgui, M., Frey, G., Li, Z., Vyatkin, V.: Designing efficient reconfigurable control systems using IEC61499 and symbolic model checking. IEEE Trans. Autom. Sci. Eng. **16**(3), 1110–1124 (2018)
15. Mubarak, H., Göhner, P.: An agent-oriented approach for self-management of industrial automation systems. In: Proceedings of 8th IEEE International Conference on Industrial Informatics, pp. 721–726. Osaka, Japan (July 2010)
16. Lepuschitz, W., Zoitl, A., Vallée, M., Merdan, M.: Toward self-reconfiguration of manufacturing systems using automation agents. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. **41**(1), 52–69 (2011)
17. Kaindl, H., Vallée, M., Arnautovic, E.: Self-representation for self-configuration and monitoring in agent-based flexible automation systems. IEEE Trans. Syst. Man Cybern. Syst. **43**(1), 164–175 (2013)
18. Strasser, T., Froschauer, R.: Autonomous application recovery in distributed intelligent automation and control systems. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. **42**(6), 1054–1070 (2012)
19. Dai, W., Dubinin, V., Christensen, J.H., Vyatkin, V., Guan, X.: Toward self-manageable and adaptive industrial cyber-physical systems with knowledge-driven autonomic service management. IEEE Trans. Industr. Inf. **13**(2), 725–736 (2017)
20. Dai, W., Riliskis, L., Wang, P., Vyatkin, V., Guan, X.: A cloud-based decision support system for self-healing in distributed automation systems using fault tree analysis. IEEE Trans. Industr. Inf. **14**(3), 989–1000 (2018)
21. Jammes, F., Smit, H.: Service-oriented paradigms in industrial automation. IEEE Trans. Industr. Inf. **1**(1), 62–70 (2005)
22. An architectural blueprint for autonomic computing. White Paper Autonomic Computing, 4th ed., IBM, Armonk, NY, USA, Jun. 2006
23. Leitão, P., Karnouskos, S., Ribeiro L., Moutis, P., Barbosa, J., Strasser, T.: Common practices for integrating industrial agents and low-level automation functions. In: Proceedings of 43rd Annual Conference of the IEEE Industrial Electronics Society, pp. 6665–6670. Beijing, China (Oct. 2017)
24. SPADE. https://pypi.org/project/spade/. Last accessed 20 Jan 2022
25. Eclipse 4diac. https://www.eclipse.org/4diac/. Last accessed 20 Jan 2022
26. Raspberry Pi. https://www.raspberrypi.org/. Last accessed 20 Jan 2022
27. Jetson Nano. https://developer.nvidia.com/embedded/jetson-nano-developer-kit. Last accessed 20 Jan 2022