IEC 61499 Distributed Design Patterns

Lisa Sonnleithner*, Bianca Wiesmayr*, Virendra Ashiwal*, Alois Zoitl*†, *Member, IEEE*
†*CDL VaSiCS*
**LIT CPS Lab, Johannes Kepler University Linz*
Linz, Austria
{lisa.sonnleithner, virendra.ashiwal, bianca.wiesmayr, alois.zoitl}@jku.at

***Abstract*—IEC 61499 emerged as a language for modeling distributed control systems. An Application is a platform-independent model that is comprised of modular components. High reusability of these components and high reconfigurability of the underlying system configuration can only be achieved with advanced design patterns. We show two example implementations of such designs and extended a pattern to foster reuse. All designs are compared based on a set of evaluation criteria. We show that our proposed design provides improved understandability and adaptability, while maintaining a slim library with highly reusable Function Block types.**

***Index Terms*—IEC 61499, Distributed control system, Software design, Software architecture, Distributed design, Design pattern, skill, choreography**

I. Introduction

Flexible manufacturing systems are required to customize products based on customer needs, thus posing new requirements on automation software. This flexibility is achieved by shifting control logic from centralized controllers to the individual stations of a production line, resulting in a distributed control system [1]. IEC 61499 [2] standardizes a language for control software that specifies a platform-independent Application model, which is linked to a hardware configuration. The software is composed from modular components and allows for a hierarchical software organisation [3]. Software design decisions are based on the adaptability for variations in the production process and the maintainability, as it is necessary to cope with the long life-cycles of production plants [4]. A decentralized architecture of the control application improves the adaptability of the software to changing hardware requirements [5]. Architectures with a central coordinator have been the state of the art in control software engineering, but introduce a single point of failure and limit the flexibility of the design [1]. In IEC 61499, the central coordinator orchestrates its subcomponents with events that can be exchanged via an adapter [6]. In contrast, a collaborative and decentralized pattern relies on the interaction between components at the same hierarchy level. This poses new challenges for resilience and fault recovery of production system, self-adaptive architectures are thus required [7]. Architectural design patterns for IEC 61499 Applications have been proposed following a centralized hierarchical approach [6] and a distributed hierarchical one [1], [5]. For the latter, the required interactions can be derived from a behavior model that captures the process steps of a production system. Possible limitations of our previous work [5] are the small example and the exact matching between tasks and components. A mechatronic component such as a robot could perform several steps within the production process. In this case it has to be triggered repeatedly.

978-1-7281-2989-1/21/$31.00 ©2021 IEEE

Nature has inspired a fully decentralized design. A group of autonomous, collaborative components of a cyber-physical system is referred to as a swarm in analogy to its equivalent in nature [8]. Within a large group of animals, such as a school of fish, each individual forms decisions that are exclusively based on local information: identified obstacles, the position of neighbours, or the location of a nearby target. The emergent behavior of the whole swarm is thus the sum of numerous decisions on local level. Following a biomimetic approach, these principles can be applied also in technical domains [9].

In this paper, we first discuss the use of choreographies in other fields and architectural approaches for IEC 61499 in Section II and describe a running example (Section III) that was first presented in [10]. In the following section, we implement the corresponding control application in a distributed hierarchical design, which we introduced in [5]. We discuss limitations of that design and propose an adaptation that addresses these problems in Section V. All control applications were implemented using the open source software Eclipse 4diac [11] as a development environment. Finally, we evaluate both designs in Section VI and compare them to a purely hierarchical implementation. We discuss advantages and disadvantages before we conclude the paper in Section VII.

II. Related Work

Software design patterns play a vital role in the overall reusability and reconfigurability of a distributed control system [12]. The concept of a "Mechatronic Object" in the domain of manufacturing machine control is introduced by Bonfè et al. [13]. According to the authors, such a mechatronic object would allow reuse if it consisted of mechanical parts, sensors, actuators, and control software routines related to the manufacturing process. In the IEC 61499 world, this mechatronic object can be mapped to a Function Block (FB) or a subapplication (SubApp), which is responsible for a certain production process or functionality.

Zoitl et al. [6] provided guidelines for building a hierarchical automation solution in IEC 61499 with a centralized top-level controller. Bonfè et al. [14] describe hierarchical design

patterns based on object-oriented modeling in the packaging industry.

Authors in [15] present a service-oriented system for reconfigurable manufacturing systems that utilizes an orchestration based coordination approach. Each component of the manufacturing system combines mechanical, electronics, and software parts that provide local control. Such a mechatronic component has its skills and functions based on a Petri Net. A Process Manager coordinates sequencing of skills and their execution based on a process description.

A skill-based engineering approach using an orchestrator as central coordinator is described in [16]. A skill is an executable function that is responsible for a process on a machine [17], [18]. Authors in [19], use generic skills of a device not only for the engineering design phase, but extend it to the runtime control of field devices. They also defined an executable vendor independent skill-metamodel in OPC UA.

In [20], the authors propose to encapsulate the functionality of a mechatronic component in an FB that then provides a service to be used within an IEC 61499 Application. In [21], the authors took a first step towards analyzing microservice architecture in industrial automation to support Plug&Produce. A miniature model of a self-organizing Plug&Produce system is introduced in [22].

Based on the enterprise service bus concept from enterprise IT, authors in [23] proposed a new concept of PLC-service bus architecture. According to the authors, such an architecture pattern improves modularity, flexibility and adaptability of software components in PLC-software.

Choreographies are defined as the high-level description of the interactions within a service-oriented system [24]. However, work on choreographies in the domain of industrial automation is scarce. A Choreography is best for a) smaller and less complex associations, and b) decentralized approaches to achieve shorter delay time [25]. Due to missing hierarchy and hiding abilities, A Choreography is not recommended for larger associations whereas Orchestration should be preferably used only in procedural associations, due to longer delay times. According to the author, a hybrid version of orchestration and choreography would be needed.

The authors of [26] analyzed strengths of a choreography in microservice-based automation architecture. According to them, choreographies allow aggregating various services more flexibly in a decentralized control to create a new independent automation function.

To improve adaptability of modular production systems, authors in [27] proposed three software patterns for a choreography of automation services. Along with that, they also identified requirements for automation service choreographies, which they are using to support the described software patterns.

III. Running Example

We now present a running example that we will use throughout the paper to illustrate the proposed design patterns, to identify potential problems, and to compare implementation

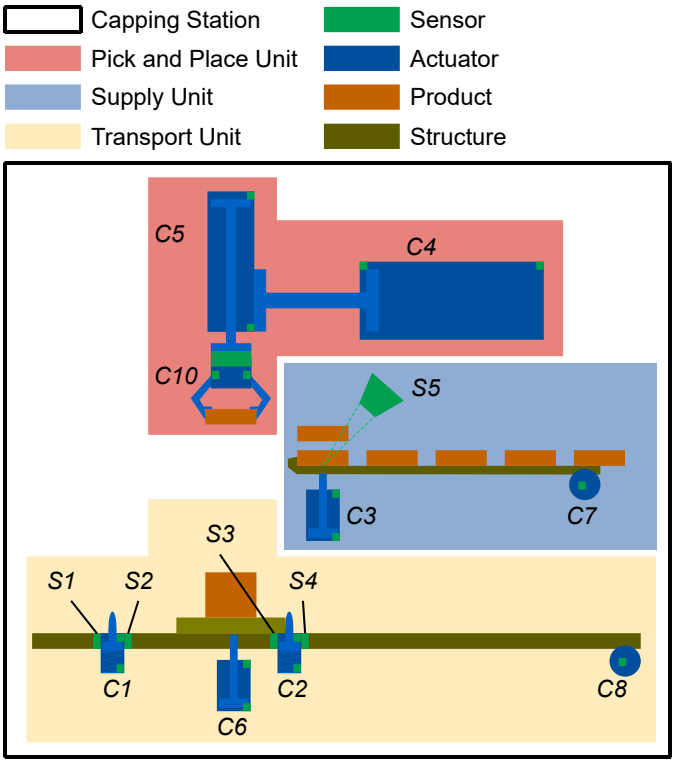


Fig. 1: Structure of the Capping Station that is composed of three main units: Supply Unit, Pick and Place Unit, and Transport Unit.

variants. We chose a capping station that was introduced in [10] and is illustrated in Figure 1. The machine places an upper part onto a lower part. It is composed of three main units, a Pick and Place Unit (PnP), a Transport Unit and a Supply Unit. Based on the description in [10], the behavior of the capping station is as follows:

The Transport Unit is responsible for letting one pallet at a time pass into the working area. This is done by the stoppers C1 and C2. Sensors located next to these stoppers detect the presence or absence of a pallet. These sensors are referred to as S1 to S4 (cf. Fig. 1). As soon as a pallet arrives in the working area, C2 stops the pallet and then the cylinder C6 locks it into position. The PnP consists of a gripper C10 and two cylinders C4 and C5. It waits above the Supply Unit until an upper part is provided by the Supply Unit. The upper part is lifted by the cylinder C3 and the PnP takes the upper part. After C3 returned to its lower position, the PnP is free to move the upper part and insert it into the lower part. A pressure sensor monitors the insertion process. After this process is successfully completed, the PnP returns to its waiting position and the pallet exits the working area to the right. A new pallet is transported into the working area and the whole process starts over. To drive the conveyors, each conveyor belt has a motor: C7 operates the conveyor in the Supply Unit, C8 the one in the Transport Unit.

This behavior is visualized in Figure 2 in an Activity Diagram that shows the sequence of actions with a focus on dependencies and parallel activities.

In the design of distributed control applications, an impor-

tant aspect is the mapping of the application to the devices of the system. To include this aspect in our investigation, we assume that each of the capping station's main units uses its own controller. The resulting system configuration is shown in Figure 3.

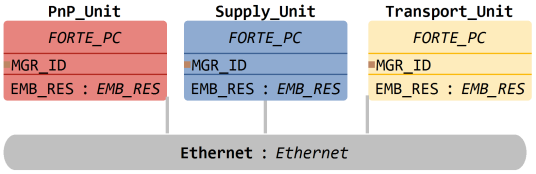


Fig. 3: System configuration of the capping station. Each unit uses its own controller that communicates with the others via a network.

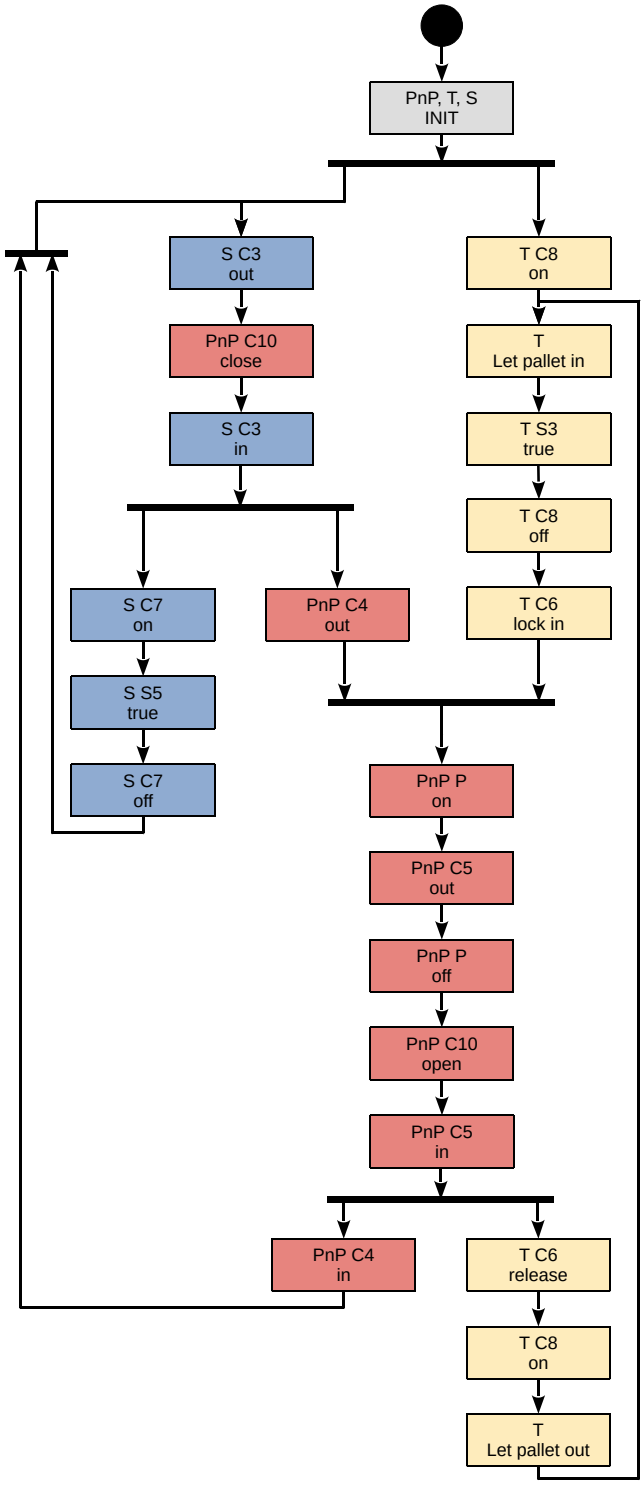


Fig. 2: Activity Diagram illustrating the sequence of individual activities (skills) of the Capping Station. Each skill is assigned to a respective unit: Supply Unit (blue), PnP (red), or Transport Unit (yellow).

IV. Distributed Hierarchical Design – Design A

Following the guidelines from [5] and [28], the substations were chosen based on the mechatronic structure: a pick-and-place unit, a supply, and a transporting component. In a distributed control system, these substations often have their own control device. Each substation is implemented in a SubApp. Within the SubApp, the developer may choose to implement an additional level of the distributed design, or to define an orchestrator. The activity diagram from Figure 2 was utilized to derive the required event signals between SubApps. An event is required for each connection between two substations, represented in the Activity Diagram as a transition between two boxes of different color. We implemented the control software of our running example according to this design. The resulting IEC 61499 Application is shown in Figure 4.

For the weigh-mix-station [5], the application resembled the structure of the Activity Diagram resulting in an easily understandable Function Block Network (FBN). In our current running example, we however observe that a single component can offer multiple functionalities, for example the PnP, both fetches a new cap from the supply, and applies it onto a lower part. As a result, the execution behavior of the application cannot be directly derived from the application.

A potential solution is to use the activity diagram, which illustrates the execution path of the application, as an artifact for further development. Potential disadvantages are an increased complexity of the language due to an additional model, and the potential to introduce inconsistencies between the application model and the behavior model. In this paper, we therefore address the aforementioned issues with a variation of the design, where each operation is mapped to one block.

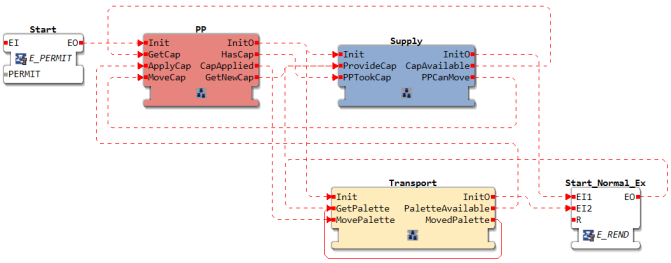


Fig. 4: IEC 61499-Application of the Distributed Hierarchical Design (Design A). Each unit is represented as a SubApp and their interactions are modeled as event connections.

V. Distributed Skill Design – Design B

To overcome the limitations that we encountered in the previous section, we take inspiration from the chain of actions pattern [12] and from orchestrated skills presented in [29].

We suggest composing a mechatronic component's IEC 61499 control software of two parts. We differentiate between services for interacting with hardware (Hardware Access FB (HW FB)) and services for interacting with higher-level components (Skill FB). These software components should be developed alongside the hardware and should allow reuse whenever a certain mechatronic component is used. The provided skills can also be used to form composite skills, i.e., skills that emerge from the composition of several basic skills. These services can be implemented as any kind of FB, such as a Basic FB or a Composite FB.

A. Hardware Access Function Block

Each mechatronic component should have a single FB that is responsible for accessing its inputs and outputs. We call this block a HW FB, see also Figure 5. This ensures that the hardware is not accidentally accessed by several FBs simultaneously and thus provides an interlocking mechanism. To offer the capabilities of a certain hardware component to the user, a defined interface should be used. This defined interface will be used by a Skill FB, which we will introduce in more detail in the next section. A Skill FB can request a certain task from the respective HW FB, which translates this request into hardware-specific I/O values. During the reading or writing process, the HW FB will be busy and unavailable to other Skill FBs. The HW FB is also responsible for ensuring that only currently executable and valid Skill requests are performed. Hence, a validation of the incoming request is required. Furthermore, detecting errors (such as lack of response from the hardware) is another responsibility of the HW FB.

B. Skill Function Block

Each mechatronic component can have one or several Skill FBs, see also Figure 5. Each Skill FB only supports one skill (such as `open`, `close`, `moveTo`, ...).

We propose two different types of Skill FBs:

- A *Basic Skill FB* is characterized by its interaction with a HW FB. Only simple tasks (e.g., setting a certain HW value) should be performed by such a block. When hardware access is required to perform the skill, the Skill FB communicates with the HW FB. Currently, we use the local multicast pattern [30], but any kind of communication is possible. Evaluating which kind of communication best fulfills the requirements of our design exceeds the scope of this paper and will be covered in future work.
- A *Composite Skill FB* is an FB that groups the interaction of several Basic Skill FBs and forms a more complicated behavior. To group Basic Skill FBs, a SubApp or a Composite Function Block (CFB) can be used. It is important to note that a Composite Skill FB does not necessarily have a single associated HW FB anymore, as it can be composed of the Basic Skills of different hardware components.

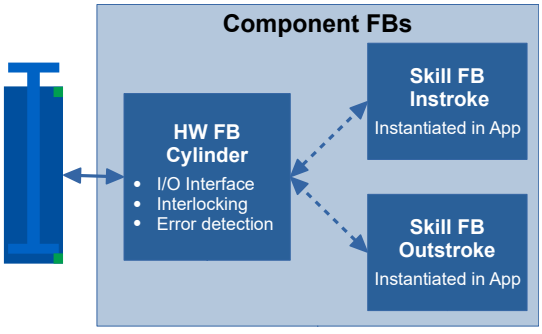


Fig. 5: Necessary FB Types that need to be developed for a cylinder. The two Skill FBs, one for the instroke and one for the outstroke, can be used within an IEC 61499 application whenever it is required for the cylinder to move in or out.

C. Combining Skill and Hardware Function Blocks

When developing the IEC 61499 software for a mechatronic component with this design pattern, then two or more FB types have to be developed (one HW FB and one or more Skill FBs). As a simple example, when developing the necessary FB types to control a cylinder, then one HW FB will be implemented that interacts with the cylinder. Additionally, two Skill FBs will be implemented, one responsible for the outstroke and another one for the instroke. This is shown schematically in Figure 5. In this figure, the connection between the HW FB and the actual hardware is drawn as a continuous line, as the HW FB directly accesses the hardware. In contrast to that, the connections between the Skill FBs and the HW FB are drawn as a dashed line. The reason for this is – as mentioned in the previous section – that these FBs are not directly connected via event or data connections, but communicate with each other via specific communication FBs. This allows a complete separation of the skills and the hardware access within the IEC 61499 application.

When developing the control software for more than one mechatronic component, Skill FB types and HW FB types could be reused (depending on the components) to maintain a slim library. Currently, this is done by configuring these FBs via interface elements. To hide these interface elements from the user, tool support for configurable FBs would be desirable.

D. Implementing Skill and Hardware Function Blocks

To further illustrate the concept of our proposed design, we will now discuss the Basic Skills that are used by the Transport Unit from Section III in detail (cf. Figure 6). The Transport unit consists of three cylinders that can move up and down, a motor that can be turned on and off, and four sensors that detect the presence or absence of pallets. In addition, the cylinders and motor have sensors to detect their current state (i.e., their position). Each of these three components requires its own HW FB Type. This FB Type is then instantiated once per hardware component, i.e., a respective instance is created for each of the four sensors, for the motor, and for each cylinder.

The Skill FBs interact with the HW FBs to allow performing the operations in various parts of the application. Accordingly, a Skill FB Type may be instantiated repeatedly in the same

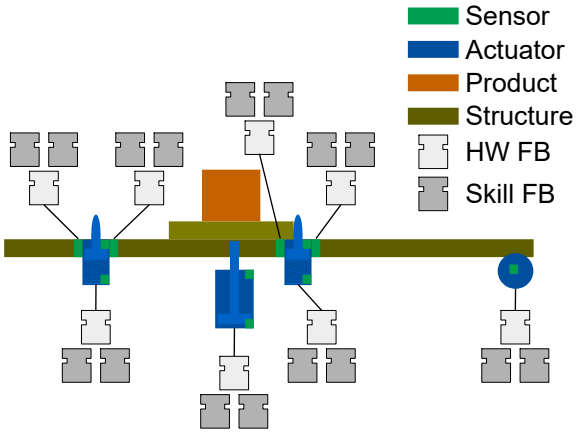


Fig. 6: A detailed view on the transport unit. Each physical component of the running example (cf. Section III) needs a HW FB and Skill FBs.

Application. The skills of the Transport unit are shown in Figure 6 and listed as follows:

- the Basic Skills `moveUp` and `moveDown`, which are offered by the cylinders `C1`, `C2`, and `C6`,
- the Basic Skills `turnOn` and `turnOff`, which are offered by the motor `C8`, and
- the Basic Skills `detectPart` and `detectNoPart`, which are offered by the sensors `S1` to `S4` in front of and behind the stoppers `C1` and `C2`

At least one Skill FB instance per hardware component (e.g., per cylinder) is required. For example, the `moveUp`-Skill FB is instantiated for each of the three cylinders. Each instance communicates with a single HW FB.

One Skill FB Type and two HW FB Types were developed to represent these skills and the hardware access. At the interface of the Skill FB Type, the skill message and the addresses of the corresponding HW FB can be set. At the interface of the HW FB Types, the addresses of the corresponding Skill FBs can be set. This configurability allowed reusing the FB Types to model all Basic Skills and the hardware access of the unit with only three different FB Types.

E. Composite Skill Function Blocks

The whole application could be modeled by only using these Basic Skills. But combining certain skills that are frequently used together to Composite Skills will lead to a cleaner application design. As an example, the Transport unit frequently uses a certain combination of Skills to let a product pass through one of the stoppers. This skill first detects whether a product is present in front of the stopper. Then the stopper moves down, and up again once the product passed the stopper, which is detected by the two sensors. The implementation of this skill is shown in Figure 7.

F. Overall Application

The same approach was used to implement all Basic and Composite Skills of the PnP unit and the Supply unit. When using these FBs in an application, the Skill FBs can be instantiated several times, whenever executing that skill is

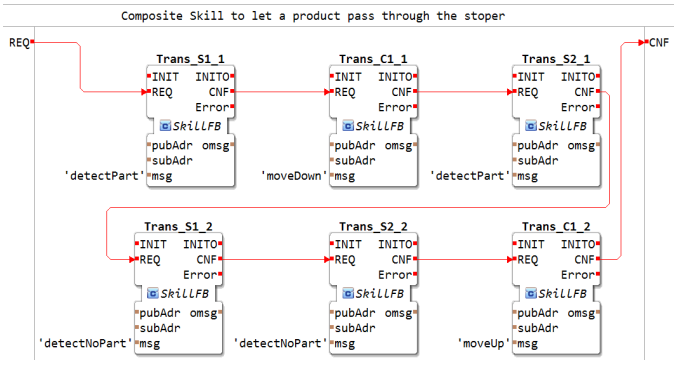


Fig. 7: Composite Skill FB that is responsible for letting a pallet pass through a stopper. It is composed of a sequence of Basic Skill FBs.

necessary. The HW FB on the other hand, is only used once in the application. Except for initialization it is not connected to other components of the application. The whole implementation of the capping station's IEC 61499 application is shown in Figure 8.

When executing IEC 61499 software with this approach, it is possible, that the HW FB is requested by several Skill FBs at the same time or by a Skill that cannot be executed at the time. The HW FB is responsible for handling and logging such situations. When testing the application, this could be an indicator for an error in the application and could therefore provide valuable feedback during the testing process of the software.

VI. Evaluation

In this section, we will compare the design that was originally proposed in [5] to its skill-based adaptation (cf. Section 8), and to a purely hierarchical design [10]. We will discuss advantages and disadvantages of the designs and compare their performance based on several evaluation criteria.

A. Unique FB Types and their Reusability

A large number of blocks that are required for a single application indicates poor reusability. A large block library increases the effort for maintenance. In Table I, we summarized the total number of developed FB types, the number of FB types used within the application (some FB types are only used within other FB types), and the total number of FB instances that were used in each of the three different implementation variants. FB types within other FB types (i.e., the content of CFBs or typed SubApps) were not counted towards the total number of FB types. In addition, we calculated how often a developed FB type was reused on average in the application, i.e., the average number of instances per FB type.

In Table I, we see that the number of developed FB types and the number of instances vary depending on the chosen design. The hierarchical design uses more individual FB types compared to Design A and Design B. Design B uses the least FB types, but the number of instances used in the application is about three times higher compared to the

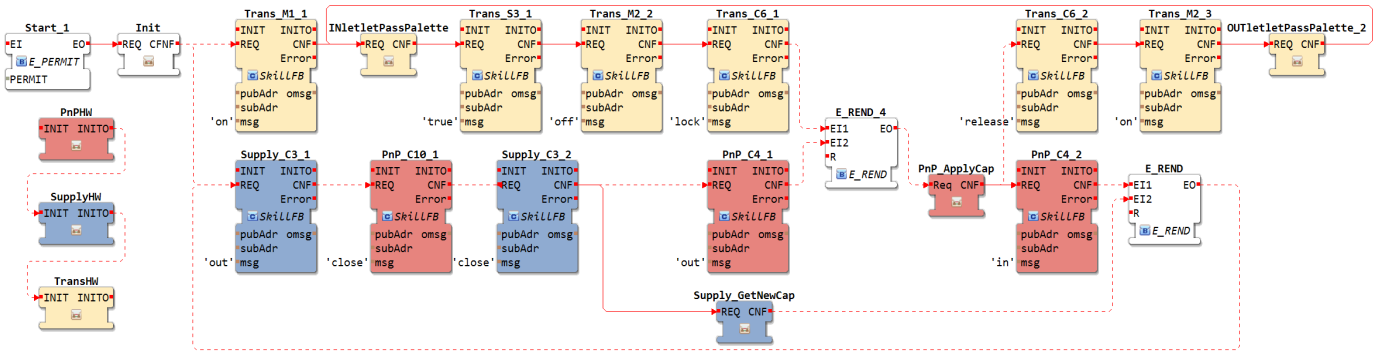


Fig. 8: Distributed Skill Design (Design B) that is composed of the interaction of Skill FBs. The HW FBs are only initialized and then not directly involved in the control flow of the application. The INIT-events for initializing the FBs are hidden to provide a better overview of the system. The structure resembles the Activity Diagram in Figure 2.

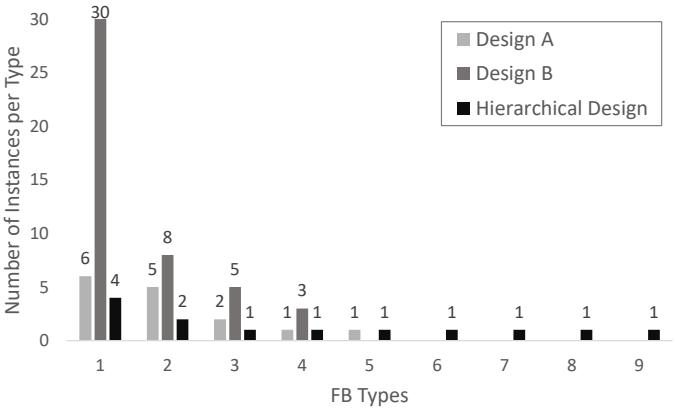


Fig. 9: A bar chart of the number of FB instances in the application of each design. The x-axis shows the developed FB types of each design, ordered from most to least number of instances per type.

other two designs. The higher number of instances, especially compared to Design A, can be explained by the separation of FBs into FBs that interact with the hardware (HW FB) and FBs that represent the functionality (Skill FB). Additionally, in Design B each time a mechatronic component should perform a certain action, a dedicated FB instance is required in the application. On the other hand, in Design A, a single FB instance can be responsible for more than one action.

In Figure 9, a bar chart illustrates how many times a FB was reused (sorted from most to least reused FB). For better comparison, this figure shows the bar chart of all designs in one diagram. The results show what can already be expected based on the data in Table I. Design B uses fewer FB types that are reused more often. In contrast to that, in the hierarchical design, only two of nine developed FB types were used more than once.

TABLE I: Comparison of the newly developed FB types (excluding adapters), the total number of FB instances within the application of those developed FB types, and the average reuse of the developed FB types of each implementation variant.

	Design A	Design B	Hierarchical
Total developed FB Types	10	8	15
Developed FB Types in App	5	4	9
Total FB Instances	15	46	13
Average reuse	3.00	11.50	1.55

B. Comparing FB Types

Metrics are used in software engineering to measure and evaluate the quality of software. For control software based on IEC 61499, a set of metrics has been proposed [31], [32]. We used the IEC 61499 Basic Function Block (BFB) Measures proposed in [32] to compare the BFB Types of the designs.

Both, the distributed hierarchical design A and the distributed skill design B, have one FB type with a significantly larger Execution Control Chart (ECC) than the other types. We used these larger types for the comparison. Within the applications, they were reused 6 and 8 times, respectively. On the other hand, there are several types with a large ECC in the hierarchical design, specifically all orchestrator FBs. Each orchestrator FB was only used once within the application. For the comparison, we used the BFB that controls a cylinder as a representative for the hierarchical design. This BFB was reused 4 times.

We can observe in Figure 10 that the hierarchical FB type's number of actions and interface elements is significantly higher compared to the distributed types'. This is because an orchestrator FB type needs to interact with the orchestrator FB type of the higher hierarchy level and with several other FBs on the lower level.

C. Mapping and Deployment

An important aspect of IEC 61499 is the mapping of applications, i.e., which part of the application will be deployed to which device. The applications of design A and B can be directly mapped, because each FB instance belongs to a certain device. The only FBs that do not belong to a certain device are synchronisation blocks or other small standard IEC 61499 library FBs. On the other hand, only the lowest hierarchy level of the hierarchical design can be directly mapped, because an orchestrator FB typically controls more than one device and therefore belongs to several devices.

D. Hidden Interfaces

A disadvantage of distributed design B is that it introduces hidden interfaces. The communication between the Skill and

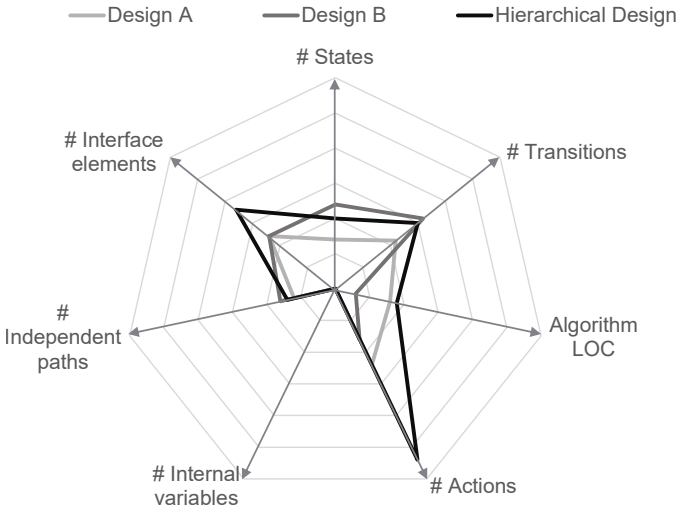


Fig. 10: *Spider Chart BFB Measure* [32] of Design A and Design B's largest BFB and of the BFB controlling a cylinder of the hierarchical implementation.

HW FBs is hidden from the user and (depending on how this communication is realized) could lead to additional communication effort. To overcome the latter, implementing tool support to automatically generate the connections between the Skill and HW FB when deploying would be a possibility. That way, the advantage of having a clean and easy to understand application would remain without introducing additional communication effort. Furthermore, tools could visualize these dependencies with different means, e.g., overlays, colors, symbols.

E. Understandability

We evaluate understandability from the perspective of someone who did not develop the application. We state the required actions for understanding it. For the application that is implemented according to distributed design A, an engineer has to trace the event connections of the application. When observing the application shown in Figure 4, we can see that this process is not straightforward, as the three units of the capping station are interconnected via several event connections. To fully understand the application, also the SubApps of each unit have to be investigated, which can be a tedious task.

To understand the application variant based on distributed design B, an engineer has to trace the events that connect various skills. It is possible to understand the application without entering any SubApps or FBs.

To understand the hierarchically implemented application, analyzing the application itself only allows to understand the hierarchical structure of the system. To understand what the application is doing, each orchestration FB has to be opened and analyzed.

F. Adaptability

When the behavior of the capping station needs to be adapted, the IEC 61499 Application needs to be changed. The extent of this change is greatly influenced by the chosen design as we showed in [5]. We will first summarize what steps are needed for each design to implement a change that does not involve new hardware. For the distributed design A and B, these steps are:

1) The application needs to be analyzed to find the correct location of the change.
2) The appropriate FB needs to be inserted.
3) The FB needs to be connected.

For the hierarchical design these steps are:

1) The application needs to be analyzed to find the correct location of the change.
2) The orchestrator FB needs to be edited.

Although the same steps are needed for distributed designs A and B, we assume that carrying out these steps will take longer for distributed design A, because understanding this application is more difficult as we discussed in Section VI-E. In general, developing or editing FB types causes additional testing effort which will increase the overall development effort of an application. Choosing an application design that allows changes without editing FB types is therefore beneficial. If also new hardware is added, then additional FB types controlling the hardware have to be developed for each design.

G. Industrial Usage

In industry, oftentimes library development is managed by a platform team and application design is managed by a project execution team. In some cases, platform development and project execution might even be managed by different companies (e.g., when company A buys hardware and software from company B and then uses these components). This means, that the person who develops an application might have limited access to FB types due to various reasons.

Depending on which design was chosen for an application, this can lead to extensive consequences in terms of understandability and adaptability of the application, as discussed in the previous section. Only design B allows to fully understand the behavior of the application without accessing a single FB. Depending on the detailed implementation, full understandability is also possible for design A. Understanding a hierarchically implemented application is only possible, if all FBs can be accessed. The same problem also arises in terms of adaptability. When a quick change to the application is needed (e.g., a bug was discovered, a hardware component needs to be changed or the process of the machine needs to be adapted) it is crucial whether this change can be implemented without having to edit a FB type.

Applications following Design B can be easily understood and edited by an application engineer, even if access rights are limited. This not only allows changes to be implemented faster, but also provides an easy way for a company to protect its intellectual property (e.g., the detailed content of FB types). Therefore, Design B could serve as an enabler for a library economy.

VII. CONCLUSION

In this work, we applied the IEC 61499 Application design that we proposed in [5] to a more complex example.

We identified challenges and proposed an adapted design in Section V. We then compared these two designs and a purely hierarchical implementation of the example. The results showed that the adapted design is promising. It provides improved understandability and adaptability compared to other designs. It also fosters reuse of FB types and can therefore reduce development effort. Due to its easy adaptability, Design B not only allows to fully implement a choreography, but it is also possible to connect the Skill FBs to an orchestrator and implement an orchestration. This means, that with Design B a choreography, an orchestration or a hybrid implementation is easily possible.

In future work, we will investigate how error handling can be implemented in our proposed design, how we might implement tool supported configurability to further reduce the number of interface elements of FBs and investigate for which use cases a choreography, an orchestration, or a hybrid implementation is better suited.

ACKNOWLEDGEMENTS

This work has received funding from the European Union's 1-SWARM project under grant agreement 871743.

REFERENCES

[1] D. Drozdov, U. D. Atmojo, C. Pang, S. Patil *et al.*, "Utilizing software design patterns in product-driven manufacturing system: A case study," in *Service Oriented, Holonic and Multi-agent Manufacturing Systems for Industry of the Future*, T. Borangiu, D. Trentesaux, P. Leitão, A. Giret Boggino *et al.*, Eds. Cham: Springer, 2020, vol. 853, pp. 301–312.

[2] IEC TC65/WG6, "IEC 61499-1, function blocks - part 1: architecture v2.0: Edition 2.0," Geneva. [Online]. Available: www.iec.ch

[3] A. Zoitl and R. W. Lewis, *Modelling control systems using IEC 61499*, 2nd ed., ser. IET Control engineering series. London: IET, 2014, vol. 95.

[4] B. Vogel-Heuser, A. Fay, I. Schaefer, and M. Tichy, "Evolution of software in automated production systems: Challenges and research directions," *Journal of Systems and Software*, vol. 110, pp. 54–84, 2015.

[5] B. Wiesmayr, L. Sonnleithner, and A. Zoitl, "Structuring distributed control applications for adaptability," in *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*, vol. 1, 2020, pp. 236–241.

[6] A. Zoitl and H. Prähofer, "Guidelines and patterns for building hierarchical automation solutions in the IEC 61499 modeling language," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, 2013.

[7] M. S. Haque, D. Jun Xian Ng, A. Easwaran, and K. Thangamariappan, "Contract-based hierarchical resilience management for cyber-physical systems," *Computer*, vol. 51, no. 11, pp. 56–65, 2018.

[8] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, "Swarm robotics: a review from the swarm engineering perspective," *Swarm Intelligence*, vol. 7, no. 1, pp. 1–41, 2013.

[9] J. M. Benyus, *Biomimicry: Innovation inspired by nature*, repr ed. New York, N.Y.: Harper Perennial, 2008.

[10] A. Zoitl, T. Strasser, and G. Ebenhofer, "Developing modular reusable iec 61499 control applications with 4diac," in *Proceedings*. Piscataway, NJ: IEEE, 2013, pp. 358–363.

[11] Eclipse 4diac, "Eclipse 4diac - the open source environment for distributed industrial automation and control systems," 2020. [Online]. Available: https://www.eclipse.org/4diac

[12] S. Patil, D. Drozdov, and V. Vyatkin, "Adapting software design patterns to develop reusable IEC 61499 function block applications," in *Proceedings IEEE 16th International Conference on Industrial Informatics (INDIN)*. Piscataway, NJ: IEEE, 2018, pp. 725–732.

[13] M. Bonfe and C. Fantuzzi, "Design and verification of industrial logic controllers with uml and statecharts," in *Proceedings of 2003 IEEE Conference on Control Applications, 2003. CCA 2003.*, vol. 2, 2003, pp. 1029–1034 vol.2.

[14] "Design patterns for model-based automation software design and implementation," *Control Engineering Practice*, vol. 21, no. 11, pp. 1608–1619, 2013, advanced Software Engineering in Industrial Automation (INCOM'09).

[15] J. M. Mendes, P. Leitão, A. W. Colombo, and F. Restivo, "High-level petri nets for the process description and control in service-oriented manufacturing systems," *International Journal of Production Research*, vol. 50, no. 6, pp. 1650–1665, 2012.

[16] K. Dorofeev and A. Zoitl, "Skill-based engineering approach using opc ua programs," in *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, 2018, pp. 1098–1103.

[17] A. Köcher, C. Hildebrandt, B. Caesar, J. Bakakeu *et al.*, "Automating the development of machine skills and their semantic description," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 1013–1018.

[18] F. Spitzer, R. Lindorfer, R. Froschauer, M. Hofmann *et al.*, "A generic approach for the industrial application of skill-based engineering using opc ua," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 1671–1678.

[19] P. Zimmermann, E. Axmann, B. Brandenbourger, K. Dorofeev *et al.*, "Skill-based engineering and control on field-device-level with opc ua," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019, pp. 1101–1108.

[20] C. Sunder, A. Zoitl, T. Strasser, and B. Favre-Bulle, "Intuitive control engineering for mechatronic components in distributed automation systems based on the reference model of iec 61499," in *INDIN '05. 2005 3rd IEEE International Conference on Industrial Informatics, 2005.*, 2005, pp. 50–55.

[21] A. Homay, A. Zoitl, M. d. Sousa, and M. Wollschlaeger, "A survey: Microservices architecture in advanced manufacturing systems," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, vol. 1, 2019, pp. 1165–1168.

[22] J. Walter, K. Grüttner, and W. Nebel, "Using iec 61499 and opc-ua to implement a self-organising plug and produce system," in *The 5th International Workshop on Model-driven Robot Software Engineering (MORSE 2018)*, 2018.

[23] V. Ashiwal, A. Zoitl, and M. Konnerth, "A service bus concept for modular and adaptable plc-software," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 22–29.

[24] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara *et al.*, "Microservices: yesterday, today, and tomorrow," 2017.

[25] "Orchestration vs. choreography functional association for future automation systems," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 8268–8275, 2020, 21th IFAC World Congress.

[26] A. Stutz, A. Fay, M. Barth, and M. Maurmaier, "Choreographies in microservice-based automation architectures : Next level of flexibility for industrial cyber-physical systems," in *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*, vol. 1, 2020, pp. 411–416.

[27] A. Stutz, A. Fay, M. Barth, and M. Maurmaier, "Software patterns for the realization of automation service choreographies," in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2021, in press.

[28] B. Brandenbourger and F. Durand, "Design pattern for decomposition or aggregation of automation systems into hierarchy levels," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2018, pp. 895–901.

[29] K. Dorofeev and M. Wenger, "Evaluating skill-based control architecture for flexible automation systems," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019, pp. 1077–1084.

[30] A. Zoitl and T. Strasser, *Distributed control applications: guidelines, design patterns, and application examples with the IEC 61499*. CRC Press, 2017.

[31] P. Gsellmann, M. M. Merkumians, and G. Schitter, "Comparison of code measures of IEC 61131–3 and 61499 standards for typical automation applications," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2018, pp. 1047–1050.

[32] L. Sonnleithner and A. Zoitl, "A software measure for IEC 61499 basic function blocks," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 997–1000.