Evaluating Skill-Based Control Architecture for Flexible Automation Systems

Conference Paper · September 2019

DOI: 10.1109/ETFA.2019.8869050

2 authors:

Kirill Dorofeev
fortiss
12 PUBLICATIONS **52** CITATIONS
SEE PROFILE

Monika Wenger
ASCon Systems
40 PUBLICATIONS **243** CITATIONS
SEE PROFILE

Evaluating Skill-Based Control Architecture for Flexible Automation Systems

Kirill Dorofeev, Monika Wenger
fortiss GmbH
Munich, Germany
{dorofeev,wenger}@fortiss.org

***Abstract*—For the manufacturing of customized products, systems require easier and better adaptivity of the production plant especially in terms of the control software. In recent research projects, a skill-based architecture has been introduced and presented within an industrial demonstrator proposing the required adaptivity. Within this work, the skill-based architecture is compared to a more traditional component-based hierarchical approach to identify strength and weakness of this approach, which then enables further improvements of the skill-based architecture. The flexibility of control software can not directly be measured. Therefore execution time, modularity, readability, and reusability considering the impact of component exchanges are used as evaluation criteria. Modularity and reusability are measured by the number of needed Function Blocks (FBs) and that can also be used for other systems respectively. To evaluate readability three derived metrics from the basic Halstead Metric are used. The comparison shows that the skill-based architecture is more complex but much more flexible than the more traditional component-based hierarchical approach.**

I. Introduction

Industry 4.0-capable automation systems require more flexibility than today's plants. The traditional control architectures include component-based design architectures, object-oriented, aspect-oriented design methods, and agile development [1]. The emergent ones include the technologies aiming at an automatic composition of software functionality, providing flexibility and plug-and-produce features [1].

There are several European and German research projects, such as SkillPro [2], OPAK [3], openMOS [4], DEVEKOS [3], BaSys 4.0 [5] dealing with the challenge of defining such flexible architectures, enabling plug-and-produce at the shop-floor level. The mentioned work served as a basis for designing a skill-based architecture for the industrial automation domain. Skills are defined as executable processes offered by industrial assets forming a production system, where the device skills are matched with the process definitions to execute the production. A reference implementation of such a skill-based architecture was realized in the VDMA demonstrator, exhibited at Automatica 2018 trade fair in Munich [6]. This demonstrator presented a vendor-independent, interoperable, skill-based solution where various automation components from different vendors were brought together in a single cell to produce a customizable product.

This paper compares the skill-based architecture with the more classical hierarchical architecture. The comparison criteria are defined according to Goal Question Metric (GQM) [7] – a method to create quality models within the software development domain.

We derive the proper metrics following GQM by identifying features that differ plants for mass production from plants for mass customization. The key feature of plants for mass customization, which is required to adapt systems to customer needs, is flexibility. Production system flexibility can be divided into reusability, readability, and modularity. Additionally, we compare the architectures in terms of the production cycle execution time, as one of the most crucial characteristics in the industrial domain.

financed by Siemens AG, Corporate Technology, Research and Development for Digitalization and Automation, Future of Automation. The authors especially thank Hermann Friedrich and Hartmut Ludwig from Siemens AG, CT RDA FOA for the continuous feedback as well as Martin Melik-Merkumians and Peter Gsellmann from the Technical University of Vienna for providing values for the Halstead Metric.

II. Control Architectures

For variable automation systems, several control architectures have been designed.

A. Service Oriented Architecture

A Service Oriented Architecture (SOA) is a style of design that guides all aspects of creating and using business services throughout their lifecycle. This allows different applications to exchange data and participate in business processes regardless of the operating systems or programming languages underlying those applications [8]. Applying SOA concepts in a field of industrial automation mainly targets to assure interoperability on both syntactical and semantic level [9] by abstracting the software components, wrapping their capabilities into a generic service interface, which later can be mapped onto the needs of a corresponding service consumer.

Bringing the service-oriented paradigms on the device level enables a seamless connection of devices from different vendors, which can be seen as a great improvement of the state-of-the-art production systems [10]. Whereas the SOA definition of a service implies that it should realize some self-contained business-functionality [11], in a production system that is not always the case. Bringing a concept of service to the lowest level of granularity in a production system hierarchy, an automation component, it can be seen that a component itself does not necessarily execute any meaningful business function. However, the components constitute the

whole system, where each component capabilities are brought together and their execution forms up the production process. The following subsections describe different views on the representation of the components functionality in production systems. This is considered as a basis to apply SOA principles in the automation domain.

B. Module Type Package

Module Type Package (MTP) [12] in process automation is a way to semantically describe process modules and their functionality enabling their efficient integration in a Process Control System (PCS). The module functionality is then encapsulated and provided to the rest of the PCS as a service, following Industry 4.0 concepts [9]. The services can be parameterized and executed in a specific order to run the desired production process. In order to use the engineering information from MTP file during the runtime, the MTP can be represented in the OPC Unified Architecture (OPC UA) namespace of a module. This enables the online discovery of the functionalities, presented in a PCS.

C. Hierarchical Architecture

The hierarchical control architecture proposed by Wenger et al. [13] describes a software structure that corresponds to the hardware structure of a plant, assuming that it is composed of different components. To create a proper software structure a bottom-up approach is recommended for building the single control components. The hierarchical architecture, therefore, introduces different layers:

- layer 0: provides the hardware interface and, therefore, performs Inputs and Outputs (I/Os) access
- layer 1: provides the elementary control operation of an atomic hardware component that usually requires I/Os access
- layer 2 to n: takes over the coordination of its sub-components
- layer n+1: provides the overall process control

Each component corresponds to one single layer. Components of a lower layer are coordinated by the next upper layer.

D. The BaSys 4.0 Control Component

The research project BaSys 4.0 [5] introduces so-called BaSys 4.0 control components. A BaSys 4.0 control component can internally contain several partly orthogonal, and partly dependent state machines that allow querying the state of a component based on the chosen access rights at any time [14]. Each control component implements at least one state machine that represents the actual occupancy state of the control component. The functionality of a control component can be divided into different driving modes. A specific driving mode has to be selected before the occupancy state of the control component is changed from idle to execute. Since each BaSys 4.0 component provides the same structure a coordination component can orchestrate control components of different vendors and/or programming languages used for its implementation.

E. The IAS-Skill

The similar idea of having a generic skill interface with the same structure was the basis of the Integrated Assembly Solutions (IAS) group of the Mechanical Engineering Industry Association (Verband Deutscher Maschinen- und Anlagenbau) (VDMA). This group consists among others out of automation component suppliers that have agreed on such skill interface. This interface [6] wraps up the functionality of the components, offering their capabilities in the form of an object in the namespace of an OPC UA Server, running on top of each component. It allows, for example, to seamlessly exchange the components within a workstation if these components support the defined interface and are capable of executing the same functionality. The software part of a control application remains the same, as there is no need to change anything: the skill execution logic is the same for any component in the system. With the devices communicating with each other over OPC UA and having a generic skill interface, the IAS VDMA group has demonstrated a tentative proof of concept of using skills to create a fully-featured industrial cell [3]. This vendor-independent skill-based demonstrator that produces a customizable fidget spinner was presented at Automatica 2018 trade fair.

Generally, a skill describes an ability of a resource to implement some production process [2]. Skills represent the data exchanges and function invocations that constitute a system operation [15]. The main purpose of defining skill is to design the automation systems based on the required skills for each step of a production process, instead of identifying actual production resources. This way, the required skills can be defined very early, during a design phase, and can thus be used in the subsequent engineering steps. In addition, suppliers can focus on resources that are capable of fulfilling the required skills for the production process. This allows interchangeability of the resources within and between manufacturers and more variance in the systems' configuration [15] [16].

It is supposed that the basic cross-vendor skills are defined by a standardization committee. A manufacturer, who offers the skill-based automation components, realizes these basic skills and can additionally implement some more component-specific skills. Finally, an application engineer, the end-user, orchestrates the provided skills without caring about low-level bytes-and-bits and programs the final process sequence [6].

Component-level atomic skills can be aggregated into higher-level skills, composed out of other atomic and/or composite skills. Skill composability allows one to build up the hierarchies, where derived skills inherit the attributes of their parents.

A device that offers a skill (skill provider) should have an interface to be controlled and observed by other parties, constituting a production system (skill consumers). Such an interface should allow not only to trigger a skill, but also to monitor the status and intermediate execution results. Note that the BaSys 4.0 control component, described in the previous subsection, can be also presented in form of an IAS-Skill.

III. Evaluation Approach

To evaluate the skill-based architecture two scenarios have been realized. The first is the initial setup that realizes a pick and place application. The second scenario considers a component exchange and its impact on the implementation. For both scenarios, a skill-based implementation, as well as a hierarchical implementation, has been created within the open-source tool Eclipse 4diac[1]. 4diac implements the IEC 61499 standard that has been designed for distributed control applications and supports mechanisms for online reconfiguration. The skill-based and the hierarchical implementation are then compared according to several evaluation criteria.

A. Hardware Setup

1) Initial Setup: The initial setup is a pick and place station, as illustrated in Fig. 1. For this setup, an endless loop application has been created, where any detected workpiece at the beginning of the transport belt is transported to the middle of the station. When the workpiece is detected by the presence sensor in the middle of the transport belt, it is stopped by a pneumatic separator. A pneumatic handling unit then inserts an inlay by extending and retracting vertical and horizontal pneumatic axes. Afterwards, the workpiece is released by the separator and moved to the end of the transport belt, where it is detected by a light barrier. The moving direction of the transport belt is then changed so that the workpiece is moved back to the beginning of the belt. In the next cycle, the inlay is removed from the workpiece and is put back on the slider. This process is performed within an endless loop. The whole process is controlled by a Wago PFC200 Programmable Logic Controller (PLC).

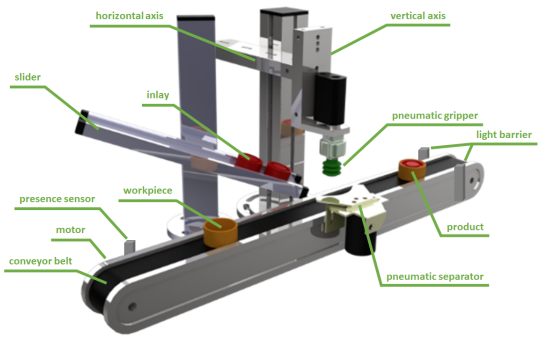

Fig. 1: Structure of a pick and place station for the initial setup.

2) Component Exchange: During the component exchange the transport belt, the motor, the separator, and the light barrier are replaced by a two-axis robot from Lego Mindstorms, as illustrated in Fig. 2. When the light barrier detects a workpiece the robot takes the workpiece and moves it to the assembly position. The robot then returns to its home position. This is required to avoid collisions since the robot and the handling unit share the working area. After the robot has reached its home position the handling unit is triggered to insert the inlay. As soon as the handling unit reached its home position the robot is moved to the middle to fetch the workpiece. The robot then takes the workpiece from the belt and rotates with it to the end position where it hits a limit switch. This corresponds to the light barrier of the initial setup. Then the robot moves back to its start position. In the next cycle, the inlay is removed from the workpiece and put back on the slider. This process is also performed within an endless loop. The robot is controlled by an EV3 controller.

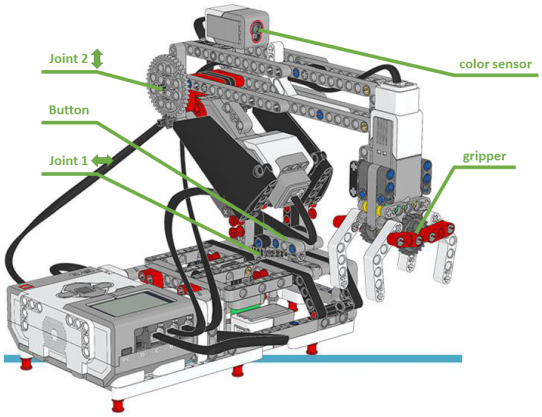

Fig. 2: Lego Mindstorms two axis robot arm [17] to replace the separator and the conveyor belt in the initial setup.

B. Control Application

1) Hierarchical Application: We utilized the hierarchical architecture described in section II-C and implemented it in IEC 61499. The component hierarchy of this application is illustrated in Fig. 3. Each of these components is controlled by one FB, where upper layer components provide a coordination FB for the corresponding lower layer components. The vertical and the horizontal axes of the handling unit are composed of two different types of pneumatic cylinders, a single acting, and a double-acting cylinder. Incoming and outgoing parts of the conveyor are counted by a part counter FB that allows detecting if the inlay has to be put into the workpiece or if it has to be taken out of the workpiece. The FBs implement the logic needed for all hierarchical levels of control and can be re-used throughout the application. For example, as the skills FBs control the system starting from layer 2, as presented in II-C, the lower level FBs for layers 0 and 1 are re-used in both applications without any change.

2) Skill-Based Application: We used the reference implementation of the IAS-Skill presented in section II-E and implement it in IEC 61499 using OPC UA as a communication mean between skill providers and consumers. In this case, the skill providers are the FBs directly controlling the I/Os, while the consumers are the FBs responsible for the high-level orchestration of the provided skills.

In our example, each of the hardware components of the pick-and-place station appears in the system with a set of

[1]https://www.eclipse.org/4diac/

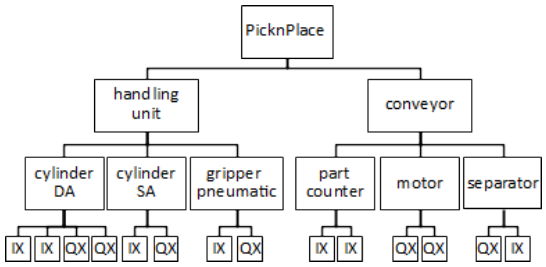

Fig. 3: Component hierarchy for the hierarchical pick and place application, according to [13], IX and QX represent the hardware inputs and outputs.

its skills. Horizontal and vertical axes can be extended or retracted, the gripper can grip or release an inlay, the separator has its skill that halts or releases a workpiece on the conveyor belt. For the conveyor belt we define the following three skills with an input parameter that sets the direction of the belt:

- $TransportThrough$ - is a skill that turns on the conveyor belt and keeps it on until the end sensor in the current direction is triggered.
- $TransportTo$ - is a skill that automatically turns on when the $TransportThrough$ is started until the separator sensor in the middle of the transport belt is triggered
- $TransportFrom$ - is a skill that automatically turns on when the workpiece is being moved from the separator until the light barrier is triggered

Each skill can be executed or stopped by calling $start()$ or $stop()$ OPC UA methods of the respective skill object. An OPC UA Client can observe the execution process by subscribing to $CurrentState$ and $LastTransition$ variables of skill that reflect the changes in the skill state machine while the process is running.

Additionally, we define next level composite skills that combine the underlying atomic skills into more abstract high-level control sequences. The $Assemble$ skill executes the sequence illustrated in Listing 1 to take a cap from the slider and put it on a workpiece.

```
1  VAxis.setDirection(down); VAxis.start();
2  Gripper.start();
3  waitFor(Gripper.currentState == Executing);
4  waitFor(VAxis.currentState == Idle);
5  VAxis.setDirection(up); VAxis.start();
6  waitFor(VAxis.currentState == Executing);
7  waitFor(VAxis.currentState == Idle);
8  HAxis.setDirection(extend); HAxis.start();
9  waitFor(HAxis.currentState == Idle);
10 VAxis.setDirection(down); VAxis.start();
11 waitFor(VAxis.currentState == Idle);
12 Gripper.stop(); VAxis.setDirection(up); VAxis.start();
13 waitFor(VAxis.currentState == Executing);
14 waitFor(VAxis.currentState == Idle);
15 HAxis.setDirection(retract); HAxis.start();
16 waitFor(HAxis.currentState == Idle);
```

Listing 1: Execution sequence of assemble skill.

$Disassemble$ skill executes a corresponding sequence for picking up the cap from the workpiece and putting it back on the slider.

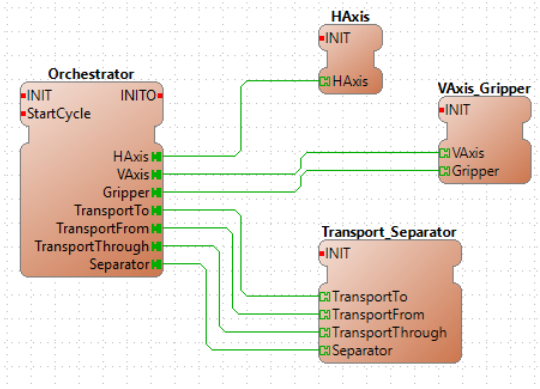

Fig. 4: 4diac skill-based application.

The overall 4diac skill-based control application is shown in Fig. 4. Every FB represents a subapplication that contains a program for a respective component, while the $Orchestrator$ FB executes the composite skills such as, for example, $Assemble$ and $Disassemble$ described above. The connections between the subapplications represent a communication channel between them, which in this case is realized as an OPC UA communication using the skill interface. The communication is done via `SClient` and `SServer` (stands for "Skill Client" and "Skill Server", respectively) FBs, which are hidden from the application in the resource view as they represent the communication blocks that can be theoretically generated out of the application without a need to program them manually. Each connection in the application uses the same interface of the same type `IAS_Skill` that allows to significantly simplify the usage of the skill interface by hiding a complex connection logic behind one adapter communication. The adapter interface is shown in Fig. 5 in a green call-out.

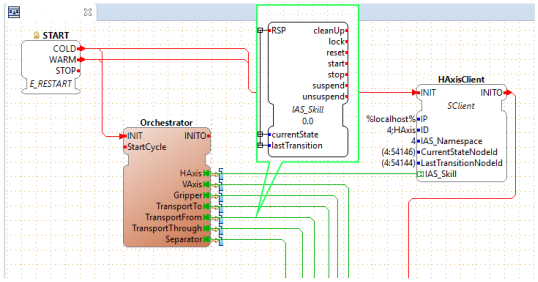

Fig. 5: Skill communication (Client side).

Each skill communication consists out of `SServer` and `SClient` FBs pair. A `SClient` FB contains the following functionality:

- Creating an OPC UA Client which is set to trigger all control methods of a skill.
- Creating an OPC UA data subscription [18], monitoring the $CurrentState$ and $LastTransition$ variables to get feedback from a server.
- `SClient` FB parameters define the following: (1) an IP

address of an OPC UA Server, where a skill is executed, (2) a skill ID contains the corresponding OPC UA namespace index for the $http://vdma.org/IAS$ namespace where the corresponding IAS-Skill Nodes are defined, and (3) the OPC UA NodeId's of the $CurrentState$ and $LastTransition$ variables so that `SClient` can subscribe to their values.

On the other side, the `IAS_Skill` adapter is connected to a FB that defines the orchestration logic of a process. In an example shown in Fig. 5 it is the $Orchestrator$ FB.

`SServer` FB:

- Creates an IAS-Skill Object in the OPC UA Server namespace. This is is an object, which implements the skill model. Every `SServer` FB creates an object of type $4diacSkillType$, which is a subtype of an abstract $SkillType$ defined by the VDMA IAS group.
- Creates a 4diac FB that contains the callbacks to each of the seven control methods of an $4diacSkillType$ instance in the OPC UA Namespace. Each time an OPC UA Client triggers a method trying to trigger a state machine transition, the corresponding FB gets this event and propagates it further to execute the low-level logic.
- As a feedback from the low-level, `SServer` FB gets an event and corresponding String values of $CurrentState$ and $LastTransition$. These values are updated each time a transition in the skill state machine has occurred.
- The `SServer` FB is parameterized by an ID (name) of a skill.

C. Evaluation Criteria

The manufacturing of customized products leads to lot size one. It, therefore, requires flexibility not only for the hardware components but also for the control software. The flexibility can be understood as the capability of a system "to be modified to support adaptation for new types of users, tasks and environments, and suitability for individualization", where modifiability is the "ease with which a system can be changed without introducing defects" [19]. The impact of component exchanges on the control application is therefore of further interest. We analyzed the required changes within the control applications after performing a component exchange scenario as described in section III-A2. Reusability of the control code is crucial for future industrial automation in order to allow the development of high-quality control systems in less time and with less cost. The costs for the development of the control software can be distributed on more plant types in case modules can be reused. The **reusability** of control applications can be measured by the number of FB that can also be used for other systems.

Features like **modularity** and **readability** provide an indication how easy a system can be modified. The entities of a control application are FBs and Program Organization Units (POUs) respectively. These entities can be interpreted as modules since they are usually supposed to run independently of each other and exchange information through their predefined interfaces.

Readability can not be measured directly. Gsellmann et al. [20] used the Halstead Metric to compare applications for the same process but implemented according to different IEC standards.

The Halstead Software Measures used in [20] are summarized in Table I. To determine readability three derived metrics from the basic Halstead Metrics can be used. The **program difficulty** describes the difficulty to read or write a program and in that case provides a measure of program understandability. It is expected that programs with a higher value are more difficult to understand. The **program effort** describes the effort that is needed to understand a program. It is expected that programs with a higher value require more effort to be understood. The **purity ratio** describes the relationship between the estimated length of a well-structured program and the total length of this program. The higher the total length of the program compared to its (estimated) well-structured program the smaller the purity ratio is. Therefore it is expected that higher purity ratio values describe a better-structured program.

For manufacturing systems, the throughput often is the most important criterion, since it determines the efficiency of a plant. The more products can be produced, the more money can be earned and the faster the break-even point of this plant can be reached. Therefore the **execution time** of architecture is also of further interest.

IV. Evaluation Results

A. Reusability

In the following subsection, we describe what changes should be made in both control applications in order to execute the component exchange scenario when one of the system components is exchanged by another device, which is capable of doing the same functionality. To execute the scenario we use the hardware setup described in subsection III-A2. We assume that the robot is pre-programmed to execute the similar task as does the conveyor belt as it is described in Section III-A2.

For a skill-based application we assume additionally that robot has a skill interface and provides the same *TransportThrough*, *TransportTo*, *TransportFrom* skills, that execute the same functionality as it is defined in III-B2.

A key stopper for automation software reuse is the current intertwining of control logic with hardware-specific code [21]. Generic I/Os configuration concept [22] shows how to use generic I/Os requirements which are mapped to the hardware-specific I/Os during the deployment. This concept, being realized in Eclipse 4diac, helps to overcome the hardware limitations as it allows to create a more reusable control code.

1) Component exchange in Hierarchical Application: To replace the transport belt by the robot arm several changes were necessary. There were also several changes in the untyped subapplication `Transport` required that are illustrated in Fig. 6. The left part of Fig. 6 shows the hierarchical application with the transport belt; the right part shows the hierarchical application with the robot arm instead of the transport belt.

TABLE I: Halstead Software Measures

Metrics	Calculation	Description
Program vocabulary size	$n = n_1 + n_2$	quantitative measure of testing difficulty and an indication of ultimate reliability, by unique number of operators n_1 and operands n_2.
Program length	$N = N_1 + N_2$	the total length of a program by the total number of operators N_1 and operands N_2
Halstead length	$\hat{N} = n_1 log_2 n_1 + n_2 log_2 n_2$	close estimate of length for well structured programs.
Halstead program volume	$V = N \cdot log_2 n$	number of bits to provide a unique designator for each of the n items in the program vocabulary.
Program difficulty to read or write a program	$D = \frac{n_1}{2} + \frac{N_2}{n_2}$	measure of program understandability.
Program effort	$E = D \cdot V$	effort to understand the program.
Purity ratio	$PR = \frac{\hat{N}}{N}$	relationship between the estimated length of a well structured program and the total length of this program.

The `Separator` and the `Motor` subapplications have been replaced by a subapplication implementing the `RobotArm`. This subapplication can be seen as a preprogrammed robot control application that is used to execute the robot pick-and-place process as it is described earlier in the subsection IV-A. This application is the same for both hierarchical and skill-based control applications. The Execution Control Chart (ECC) of the `Conveyor` Basic Function Block (BFB) has been adapted to consider the overlapping working areas of the robot arm and the handling unit. Since the I/Os on the EV3 brick currently does not support hardware-triggered indication event `IND`, a `SensorPartOut` subapplication has been created, that replaces the presence sensor of the transport belt. `SensorPartOut` subapplication is also mapped on the EV3 device. It cyclically checks the value of the robot arm limit switch and fires an event whenever the switch is triggered.

To establish a communication between EV3 and Wago PFC200 `PUBLISH` and `SUBSCRIBE` FBs have been added inside the resources of both devices, which create a User Datagram Protocol (UDP) connection between the devices. As it can be seen in Fig. 6 there are several changes required to realize the component exchange scenario. Especially the changes for the coordination component (here the `Conveyor`) can be quite time-consuming.

2) Component exchange in Skill-Based Application: The skill-based architecture was designed to handle such scenarios, therefore there is nothing to be changed in the distributed application itself. The transport subapplication that was previously deployed on the Wago PFC 200, as well as the rest of the application, is in this case exchanged by the robot control program. Only the IP addresses of the OPC UA servers and the node id's of the *TransportThrough*, *TransportTo*, *TransportFrom* skills should be changed in this scenario. The localhost connection used for the application running on WAGO PFC 200 is exchanged by the IP address of the EV3 controller. The `SClient` FBs that were previously responsible for the localhost communication on Wago PFC 200 are now parameterized with the IP address of the robot OPC UA Server and the skill node id's are adjusted correspondingly.

Furthermore, it is possible to switch the hardware that executes the *Transport* skills during runtime. In 4diac-IDE it can be done by forcing the IP-address of the desired backend device. After forcing a new IP-address, it is needed to force `INIT` event of the FB so that the `SClient` overwrites an old address with a new one. After that in the next production cycle, a new device will execute the *Transport* skill.

Skill-based engineering serves to extend the scope of reusability, introducing multiple logical levels of abstraction in a control application. Theoretically, if there are two devices offering the same skill, there is no difference in execution if the skill is triggered using one backend device or another. Thus, for example, if there is a need to reuse a transport skill, there is no difference if it is a conveyor belt or a robot, who executes a composite skill *Transport*, as well as there is no difference if, in the background, these skills execute different underlying atomic skills. Unless both, the conveyor belt and the robot, are capable of moving a workpiece from A to B, which is needed for a process, their capabilities can be reused using the same skill interface.

B. Readability

The Halstead Metric has been used as an indication for readability. This metric is calculated for the initial setup described in section III-A1 to compare the readability of both architectures. The basic metric values are calculated by the unique and total number of operators and operands. For control applications, operators and operands can be determined as described by Zhabelova et al. [23]. The values for the operators and operands have been provided by Gsellmann et al. [20], who introduced this method and created a tool to automatically determine the unique and total number of operators and operands of IEC 61499 applications. From the basic metric values the program difficulty, the program effort, and the purity ratio have been calculated, as illustrated in Table II.

Comparing the program difficulty it can be seen that the overall application of the skill-based application is only slightly more difficult to understand than the hierarchical application. This higher value is caused by a higher program difficulty of the coordinating part (`PicknPlace`) of the skill-based approach, which uses a FB-network in contrast to the BFB of the hierarchical application. Comparing the program effort it can be seen that the effort needed to understand the

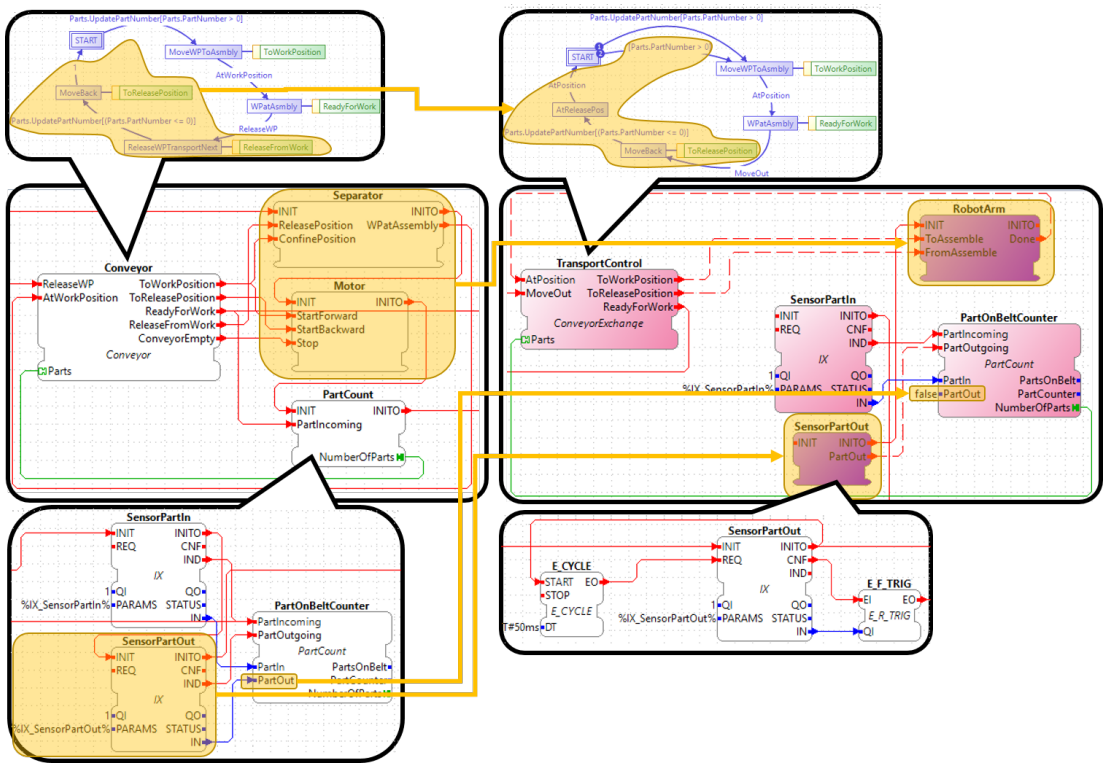

Fig. 6: Changes inside the subapplication Transport for the component exchange of the hierarchical application.

TABLE II: Halstead Metrics, hierarchical application (HA), skill-based application (SA).

	Difficulty		Effort		Purity	
	HA	SA	HA	SA	HA	SA
PicknPlace	8	29	2.334	39.803	2	2
Transport	18	11	20.368	4.344	4	1
HandlingUnit	40	33	88.500	99.527	3	2
Overall Application	65	70	281.669	398.454	4	3

overall application of the skill-based application is higher than for the hierarchical application. This difference is caused by the coordinating part (`PicknPlace`) of the skill-based approach but also by the control of the handling unit. Comparing the purity ratio the hierarchical application is only slightly better structured than the skill-based one.

C. Execution Time

Table III shows the application execution times using different control architectures. In our case, the execution time values for the skill-based application given in this sub-section highly depend on the current implementation and are highly influenced by OPC UA communication. For this comparison, we chose assembly and disassembly processes as they are the most critical ones in terms of component communications needed for the skill-based solution. At this point the overhead of the OPC UA communication is the biggest compared to the other process steps. The time is measured from the moment when the separator presence sensor is triggered as a workpiece arrives at the working position until the moment when the workpiece leaves it. There are 12(6) OPC UA method calls and 7(3) OPC UA subscriptions needed to execute a process in the skill-based applications for assembly (disassembly) process. We did 10 runs for both hierarchical and skill-based control applications. The hierarchical control application, as expected, performed faster compared to the skill-based application where the OPC UA communication brings an additional overhead.

TABLE III: Average execution time in *milliseconds* for different applications.

	Assemble	Disassemble
Hierarchical	2950	1860
Skill-based	3278	2065

D. Modularity

The number of modules influences the implementation time, the memory usage on the device and the readability. The number of FB instances defines how much memory is needed for the application to be executed on the device. The amount of FB types determines the disk space needed for the Runtime Environment (RTE). Each user-defined FB requires some amount of engineering time for the initial implementation of the application. The more FB instances within an application are used the more time is needed to implement the entire application for the plant. Table IV lists the number of modules used within the different control application architectures.

FB instances inside of Composite Function Blocks (CFBs) instances are only counted once since they only influence the implementation time when the FB type is created.

TABLE IV: Module measurements of the control application types, function block types (FBT), user-defined (UD), function block instances (FBI).

	Number of		
Application type	**FBT**	**UD FBT**	**FBI**
Hierarchical	12	9	27
Skill-Based	31	11	146

User-defined FB types can be divided into two groups, architecture specific FB types and plant specific FB types. Architecture-specific FB types are only created once and therefore only influence the initial implementation time. Plant specific FB types are created for a specific plant component that they can only be used for similar components. The hierarchical architecture does not use any architecture-specific user-defined FB types, whereas seven of eleven user-defined FB types of the skill-based architecture are architecture-specific. The higher number of used FB types and FB instances within the skill-based architecture is not only caused by the additional architecture specific FBs but also by the use of FB types from the standard library instead of plant-specific FB types.

V. Conclusion

In the paper, we evaluated the skill-based control architecture and compared it to the more traditional hierarchical architecture. Even though the skill-based solution is a novel implementation, based on a non-real-time OPC UA Client-Server communication, its performance is comparably high, bringing not a big overhead in comparison to the traditional approach. The applicability of the skill-based architecture was evaluated by implementing an industrial robotic cell using the approach [3] [6]. The further developments towards implementing OPC UA Pub/Sub mechanism as well as utilizing Time Sensetive Networking (TSN) can lead to higher performance, making the skill-based application real-time capable. Despite the fact that according to Halstead metrics provided in the paper, the skill-based application is more complex both in implementation and readability, these characteristics, however, can be very much improved in the next development iterations by re-designing the skill FBs and improving the service modeling in IEC 61499. The skill-based techniques showed their strength in the component exchange scenario, where its advantage over the traditional approach is noticeable. In the reconfigurable production environments, the skill-based architecture can gain the overall operational time due to its reusability and fast change-over process.

References

[1] V. Vyatkin, "Software engineering in industrial automation: State-of-the-art review," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1234–1249, Aug 2013.

[2] J. Pfrommer, D. Stog, K. Aleksandrov, S. E. Navarro, B. Hein, and J. Beyerer, "Plug & Produce by Modelling Skills and Service-Oriented Orchestration of Reconfigurable Manufacturing Systems," *Automatisierungstechnik*, vol. 63(10), pp. 790–800, 2015.

[3] B. Brandenbourger and F. Durand, "Design Pattern for Decomposition or Aggregation of Automation Systems into Hierarchy Levels," in *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2018.

[4] P. Danny, P. Ferreira, K. Dorofeev, and N. Lohse, "An Event-Based AutomationML Model for the Process Execution of Plug-and-Produce Assembly Systems," 2018.

[5] S. Malakuti, J. Bock, M. Weser, P. Venet, P. Zimmermann, M. Wiegand, J. Grothoff, C. Wagner, and A. Bayha, "Challenges in Skill-based Engineering of Industrial Automation Systems," in *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2018.

[6] P. Zimmermann, E. Axmann, B. Brandenbourger, K. Dorofeev, A. Mankowski, and P. Zanini, "Skill-based Engineering and Control on Field-Device-Level with OPC UA," in *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019.

[7] R. van Solingen and E. Berghout, Eds., *The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development*. McGraw-Hill, 1999.

[8] E. Newcomer and G. Lomow, *Understanding SOA with Web Services*. Addison-Wesley, 2005.

[9] T. Usländer and U. Epple, "Reference Model of Industrie 4.0 Service Architectures," *Automatisierungstechnik*, vol. 63(10), pp. 858–866, 2015.

[10] F. Jammes and H. Smit, "Service-oriented paradigms in industrial automation," *IEEE Transactions on Industrial Informatics*, vol. 1, pp. 62–70, 2005.

[11] N. M. Josuttis, *SOA in Practice. The Art of Distributed System Design*, 1st ed. OReilly Media, Inc, 2007.

[12] J. Bernshausen, A. Haller, T. Holm, M. Hoernicke, M. Obst, and J. Ladiges, "Namur Modul Type Package Definition," *atp magazin*, vol. 58, no. 01-02, pp. 72–81, 2016. [Online]. Available: http://ojs.di-verlag.de/index.php/atp_edition/article/view/554

[13] *Distributed Control Applications: Guidelines, Design Patterns, and Application Examples with the IEC 61499*. CRC Press, 2017, ch. Hierarchically Structured Control Application for Pick and Place Station.

[14] C. W. Ulrich Epple, Julian Grothoff, "BaSys 4.0: Metamodell der Komponenten und ihres Aufbaus," 2018.

[15] K. Dorofeev and A. Zoitl, "Skill-based Engineering Approach using OPC UA Programs," in *IEEE International Conference of Industrial Informatics (INDIN)*, 2018.

[16] S. Profanter, A. Breitkreuz, M. Rickert, and A. Knoll, "A Hardware-Agnostic OPC UA Skill Model for Robot Manipulators and Tools," in *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019.

[17] "Lego Mindstorms Construction Manuals," https://education.lego.com/de-de/support/mindstorms-ev3/building-instructions, accessed: 2018-09-12.

[18] "OPC UA Specification Part 4: Services. Release 1.04," https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-4-services/, accessed: 2019-03-19.

[19] ISO/IEC/IEEE, "ISO/IEC/IEEE 24765: Systems and software engineering - Vocabulary," 2017.

[20] P. Gsellmann, M. Melik-Merkumians, and G. Schitter, "Comparison of Code Measures of IEC 61131-3 and 61499 standards for Typical Automation Applications," in *International IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, 2018.

[21] I. Hegny, T. Strasser, M. Melik-Merkumians, M. Wenger, and A. Zoitl, "Towards an increased reusability of distributed control applications modeled in iec 61499," in *IEEE 17th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2012.

[22] W. Eisenmenger, J. Mesmer, M. Wenger, and A. Zoitl, "Increasing control application reusability through generic device configuration model," 09 2017, pp. 1–8.

[23] G. Zhabelova and V. Vyatkin, "Towards software metrics for evaluating quality of IEC 61499 automation software," in *IEEE Conference on Emerging Technologies Factory Automation (ETFA)*, 2015.