Skill-based Engineering Approach using OPC UA Programs

Kirill Dorofeev
fortiss GmbH.
Guerickestr. 25
80805, Munich, Germany
dorofeev@fortiss.org

Alois Zoitl
fortiss GmbH.
Guerickestr. 25
80805, Munich, Germany
zoitl@fortiss.org
Johannes Kepler University
Altenbergstr. 69
4040, Linz, Austria

***Abstract*—The modern manufacturing and production systems are changing into a flexible factory layout that raises a demand for distributed control systems, where various components from different vendors communicating over their capabilities (skills) to execute a production task. This paper offers a way of how the skills, offered by an automation system component, can be represented using a generic interface to allow the higher control levels to easily orchestrate the skills available in the system and control their execution in the runtime. We show how a skill can be modeled using a Finite State Machine and present a concrete example how the concept can be realized by means of the OPC UA communication protocol.**

***Index Terms*—OPC UA Programs, Skill-based Engineering, Plug-and-Produce, State Machines**

I. Introduction

The ongoing manufacturing systems transformation from mass production towards mass customization, coming up to the extremes of batch size one production, requires more flexible engineering solutions than the existing ones. Currently, most of the manufacturing systems are inflexible, being mainly centralized, which does not allow to easily reconfigure an existing production line to produce different product variants. Nowadays the traditional industrial control systems are reaching the limit of programmability [1] [2]. The issue is caused by the steadily growing complexity of the systems [3]. The size of such systems is limited to approximately 100000 I/O points [4]. A batch size one production raises a demand for distributed control systems [5], where multiple devices from various vendors are controlling different types of objects [6]. These systems should provide flexible solutions, capable of dealing among others with various processes, processes variations, dynamic rescheduling and rerouting, and machine reconfiguration.

Nowadays most control application are based on the International Electrotechnical Commission (IEC) 61131 standard. It was developed for centralized systems, which is reflected by its language structure, cyclic execution scheme and the support of global variables [7]. The entire language structure is defined within part three of the standard [8]. The IEC 61499 standard [9], published in 2005, has been developed for distributed architectures. This standard is able to handle the complexity of connected automation systems and also to significantly reduce the development time by overcoming signal mapping and driver programming for communication between different devices [10]. Alternatively to IEC 61131, where an application is executed cyclically, IEC 61499 introduces a concept of events to explicitly define the execution logic of a distributed system. Moreover, IEC 61499 function blocks can be reused on any platform, thus offering more flexibility compared to IEC 61131. IEC 61499 uses eXtensible Markup Language as a storage and exchange format, which enables the migration between different IEC 61499 software tools and allows the development of platform-independent PLC code [7]. Its flexibility, reconfigurability and reusability makes IEC 61499 a target technology to handle Industry 4.0 automation systems.

We utilize a concept of skill [11] [12] for wrapping the back-end functionality, needed for a production execution, into the structure that is independent from the implementation. We show how a service (skill), realized either in IEC 61131-3 or in IEC 61499, using the same interface can be exposed by the means of OPC Unified Architecture (OPC UA) [19]. A common interface for the services enables easy communication and control of the service execution. Using OPC UA Programs state machine as a common base for the service realization allows the high-level control systems to easily browse the services offered by the back-end and to trigger the service sequences needed for the production. A common interface independent from the service implementation enables interoperability between different programming standards and engineering tools. This allows to build a system upon the hierarchical skill model, where the skills scale from the atomic (basic) service, e.g., open a valve or read a sensor value, up to complex composite skills that can have different combinations of other composite and atomic skills.

Note that in this paper we use terms *service* and *skill* as equivalent terms.

The rest of the paper is organized as follows. After giving a related work overview in Section II, Section III describes how an industrial service (a machine skill) can be mapped to the OPC UA Programs and gives an example of how the Pack-

ML State Machine, wide-spread in the batch industry, can be realized in this context. Section IV describes a small evaluation example, showing a small application where two skills are orchestrated using the proposed concept. We summarize possible further investigation steps and conclude in Section V.

II. State of the Art

A. Service-Oriented Architecture in Automation Domain

Service-Oriented Architecture (SOA) was originally developed for the organization of business processes. SOA is a paradigm for organizing and utilizing complex distributed systems. The three key concepts in SOA are services, interoperability and loose coupling [13]. A possible application of SOA concepts in industrial automation was proposed in [14]. Furthermore, a possible application of two general SOA reference models: the OASIS (Organization for the Advancement of Structured Information Standards) model [15] and the SOCRADES (Service-oriented cross-layer infrastructure for distributed smart embedded devices) model [16] in the automation domain is shown in [17]. The web services are utilized here as a natural application of an existing web technologies in the automation control systems. [18] also investigates a concept of microservices to implement a SOA design for an automation system.

However, to improve acceptance rate it is crucial to demonstrate how SOA could be implemented based on standard communication accepted in the automation domain, like OPC UA [19], a wide-spread family of technologies intended to provide a communication between field devices and upper control levels. OPC UA is combined with both IEC 61131-3 [20] and IEC 61499 function blocks [21], which are two domain specific programming languages used to program the automation control systems. Thus, the SOA can be build via realizing the services implementation in either IEC 61131-3 or IEC 61499 and exposing those by means of OPC UA [22] [12].

In [23] it is shown, how the PLC programming can be simplified by wrapping the desired actions into high-level abstractions. [24] demonstrates a SOA-based approach for batch process programming and shows an implementation example using IEC61499 for the service implementation.

Bringing it together, we show how a skill being implemented either in IEC 61131-3 or in IEC 61499, can be mapped to a common OPC UA interface to enable a SOA-oriented design for a Plug& Produce [25] assembly system.

B. OPC UA Programs

OPC UA Programs [26] can be seen as a way to achieve interoperability between tools, runtimes and different standardized ways used for the service (skill) implementation. Using Programs allows to expose, discover and call these functions in a general manner. They provide a normalizing mechanism for the semantic description, invocation, and reporting result of the server functionality. According to the OPC UA Specification, Programs should model complex and state full long-running functionality in the system. In OPC UA, Programs are intended to represent a complex functionality in an OPC UA Server, for example, to run and control a batch process, execute a machine tool part program, or manage a domain download.

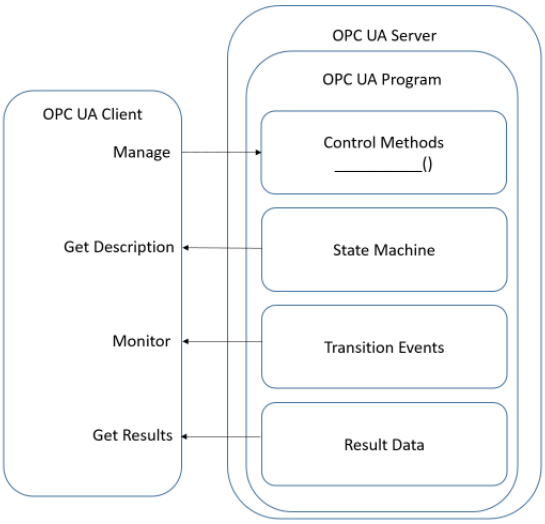


Fig. 1. Structure of an OPC UA Program

The OPC UA Clients can not only trigger the execution and get the result back but also monitor the intermediate results while the process is running. This also includes a possibility to intervene an execution to make some modifications in the runtime. An execution of an OPC UA Program is defined by a Finite State Machine. The default state machine defined in the OPC UA Specification must have four mandatory states, nine mandatory transitions between those states, and five optional methods to allow a Client to control the execution as well as a Result Data output to get the information about the program execution. The latter output is used for getting the final parameters after the program execution is terminated, e.g., power consumed while running a service, as well as intermediate data, while the program is running, e.g., percentage of completeness. An overview of a Program interface is given in Figure 1. The standard OPC UA Programs finite state machine is depicted in Figure 2. All Programs instances must support the base set of states depicted. An OPC UA Client, while viewing a Program, can observe a Program's current state and the last transition happened in the state machine. Programs do not necessarily require any Client action to execute. Unless control methods, which all are optional for each Program instance, are not defined, all Program transitions can be modeled as internal ones. In this case a Program models some process, internal for an OPC UA Server, and an OPC UA Client just gets a notification on any transition happened.

While executing the methods controlling the state machine, each state transition fires an event, which can be bound with a data, representing intermediate execution as well as a final output. OPC UA Programs can describe various state machines, as they allow to define infinite number of sub states for the mandatory four super-states, allowing in that way to map the existing state machines as OPC UA Programs and to be implemented in the OPC UA server. Overall, OPC UA Programs present a flexible mechanism, which allows to model

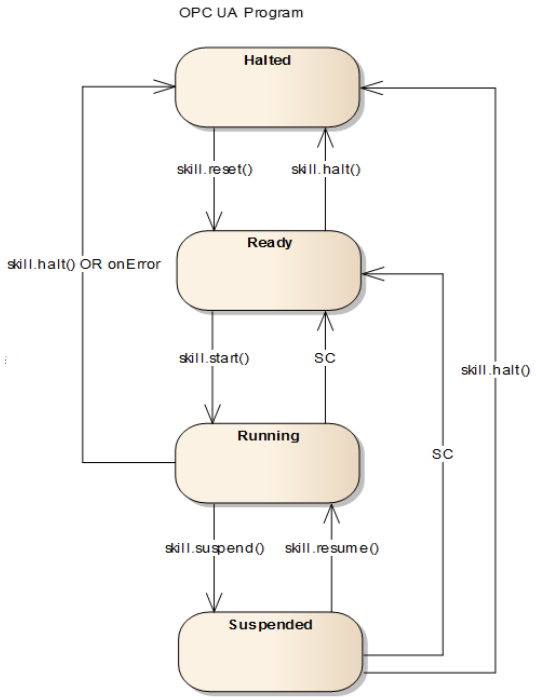


Fig. 2. A Skill Modeled as an OPC UA Program Finite State Machine

long-running processes. Unlike OPC UA Programs, the OPC UA Methods by the OPC UA Specification have a time-out of about 10 seconds, within which the method must return a result. Moreover, the Programs allow to get the intermediate data, showing the details of the execution in the runtime. Based on that data, a Client can make the decisions about further processing. All that makes the OPC UA Programs suitable for the skills modeling of a production machine, offering a flexible interface, which can be used for modeling services of different complexity. For an atomic skill, e.g., open/close valve, one can implement only two methods: *start()* and *stop()*, whereas a complex composite skill, may require additional methods as well as introducing additional sub states that can be added to the four OPC UA Programs super-states.

Currently, there is almost no support for OPC UA Programs in the existing stacks [27] [28] [29] [30], as they lack some features to build up Programs. In Section III-B we show how we model the skill in such a way that it can be implemented using the current OPC UA features available in the stacks, with an outlook on how the skill implementation can be done in the future using the OPC UA Programs.

III. Concept

A. Skill State Machine

Skills represent the data exchanges and function invocation that constitute a system operation. A skill can scale from a very primitive functionality, atomic skill, (e.g., *open/close a valve*, *read a sensor value*) to composite skills that are a combination of atomic and/or other composite skills (e.g., *pick and place*, *pack a product*). What should be defined, however, is a general skill interface, so that every system that needs to execute a skill for completing its tasks knows how to control each skill in a general way and can easily build the skill sequences out of the skills, available at the current moment. Thus, a Client first matches the product configuration, given by the higher control levels (e.g., Human Machine Interfaces, cell controllers, or other supervisory control and data acquisition type systems), with the services, offered by the assembly system. As a second step, Client orchestrates those services to fulfill production task, e.g., chooses the subset of skills relevant for a task, puts them in order.

A skill state machine is a description of a skill that can be observed from the outside of a system component (a workstation or a device, involved in the production). It hides away the skill implementation, offering the monitoring information needed, e.g. current skill state, and control ports to influence the skill execution, e.g. start/stop/pause a skill. To model an atomic skill, it is enough to have a simple state machine that realizes *Idle*, *Running*, *Pause/Stop* and *Error* states [31], that represent a component ready for execution, running a task or having an error, respectively.

For modeling more complex skills for the whole workstations, it is often needed to build up more complex state machines. For example, the PackML state machine [32] is used in packaging industry to standardize the access to machines functionality and to collect the machine key performance indicators, such as Overall Equipment Effectiveness, in a standardized manner. Using the PackML state machine in application software for the batch industry controllers, makes the software more efficient, flexible and reusable. The PackML state machine consists of seventeen states, that constitute a standard, generic way of how machine operates. It has six stable states (*Stopped*, *Idle*, *Execute*, *Suspended*, *Held*, *Complete* and *Aborted*) that can be valid for a long period of time, as well as eleven ”-ing” states (*Stopping*, *Resetting*, *Starting*, *Suspending*, *Un-Suspending*, *Holding*, *Un-Holding*, *Completing*, *Aborting*, *Clearing*) that are valid only for a certain period of time and are switched to the stable states without operator intervention. All machines that follow PackML state diagram have a state describing production (*Execute*), stoppages (*Stopped*, *Idle*) and error detection (*Aborted*). Suspended state represents the use-cases when the machine waits for the upstream (lack of material) or downstream (blocked) process to finish before it continues its production. Hold state represents operator actions that interrupt the machine execution (pause function, for example). The methods (*Reset, Start, Suspend, Un-Suspend, Hold, Un-Hold, Stop, Abort*) allow operator to control the production.

PackML state machine can be mapped to OPC UA Programs state machine, where the pure PackML states are modeled as sub-states of the OPC UA state machine. Here we propose one possible mapping, whereas one can argue about which PackML sub states should belong to one or another OPC UA super-states. In our model we follow the following constraints:

- The transitions between super-states should happen only on triggering the PackML methods

- There should be at least one Pack ML stable state as a substate of each super-state in the OPC UA Programs state machine

Considering the semantics of the states we offer the following mapping:

- OPC UA Programs Halted state includes two Pack ML stable states Aborted and Stopped as well as Aborting, Clearing and Stopping states. The Clear method leads to switch between Aborted and Stopped sub states inside the Halted super state. The Abort method leads to the Aborting state from each and every sub states in the state machine following the Pack ML State machine. The Stop method brings the state machine from every sub state of Runnning and Suspended super states to the Stopping sub state of Halted super state. These two methods correspond to the Halt method in OPC UA Programs state machine.
- OPC UA Programs Ready state includes Pack ML Idle and Resetting states. PackML Reset method maps one to one to the OPC UA Programs Reset method.
- OPC UA Programs Running state includes Starting, Execute, Un-Holding, Un-Suspending, Completing, and Complete states of PackML state machine. The Hold and Suspend methods trigger the transitions to corresponding Holding and Suspending sub states of the Suspended super state, described later. These two methods correspond to the OPC UA Programs Suspend method.
- OPC UA Programs Suspended state includes Holding, Held, Suspending and Suspended states of the PackML state machine. As OPC UA state machine has only one state that describes the pausing of the machine, whereas PackML differ between Held and Suspended. We model these two Pack ML states as two sub-states of the OPC UA Programs Suspended super state. The Un-Suspend and Un-Hold methods can be seen as OPC UA Resume method.

The resulting state machine mapping is shown in Figure 3

B. Modeling a Component Skill

A machine skill being modeled as OPC UA Program offers a client to get the description of its current state via skill state machine. At every moment of time, a Client can observe a skill state machine, get its current state and the last transition, as well as trigger necessary control methods to execute production. A Server, running a skill, can also use this state machine, for example, for preventing Client to execute methods, not allowed in the current state. E.g., the skill state machine will deny *start()* method call automatically if a skill is not in *Ready* state. Thus, it can be seen as a contract between a Client and a Server, which dynamically constitutes the skill execution and further commands, which a relevant at the moment.

A skill mapped to OPC UA from the Server side point of view can be seen as a function block that has transition events and information about a skill's current state as its inputs and control methods as outputs. A Server receives from a Client a control method call and sends back transition events of a skill state machine as they happen on the Server side. A possible skill function block realized in IEC 61499 and its representation in OPC UA namespace that uses a skill state machine depicted in Figure 2 is shown in Figure 4.

For the approach validation we use Eclipse 4diac [33], the open-source IEC 61499-compliant IDE and runtime environment. As the open62541 OPC UA stack, used in 4diac as OPC UA implementation, does not yet have OPC UA Programs support, we use method calls for Client to trigger a skill and OPC UA Data Subscription to get a feedback from a Server about transition events happened. Note that in a pure OPC UA Programs, one should use OPC UA Events to get this transition notifications back. However, a transfer from OPC UA Data Subscriptions to Events could be done seamlessly as soon as Events will be supported in the open62541 OPC UA stack [28]. Next step will be an implementation of a skill as an OPC UA Program.

IV. Evaluation

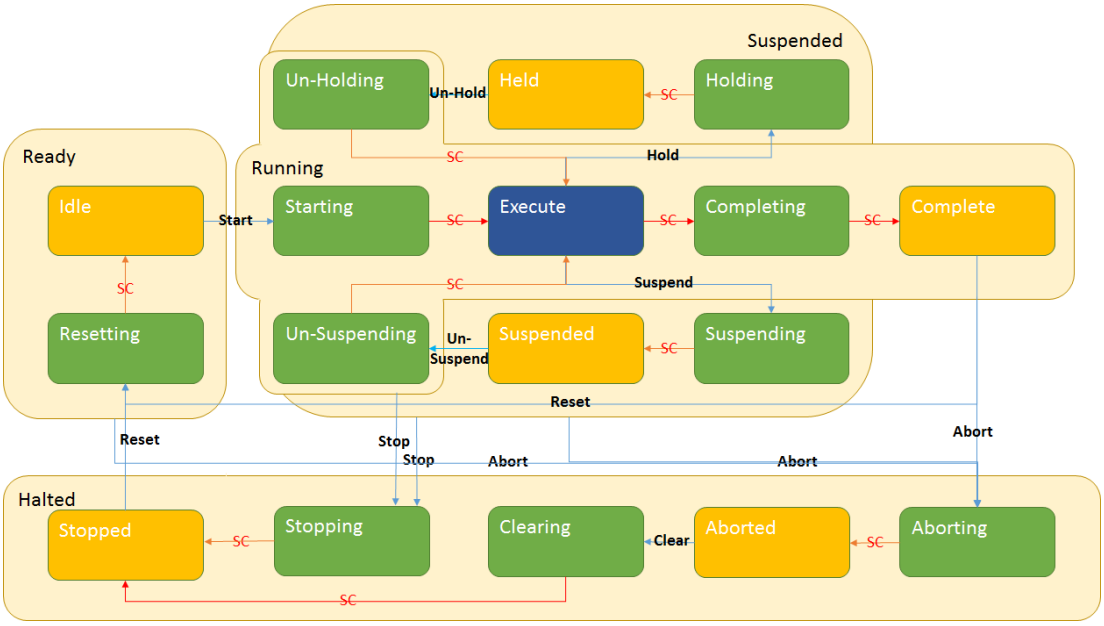
In the following example we show how using a skill concept being mapped to the OPC UA Server namespace can provide interoperability on the hardware and software levels. The demo application, modeled in Eclipse 4diac, is shown in Figure 5. There is an easy setup with three lamps - red, yellow, and green - each controlled by three different PLC's: Wago PFC200[1] and RevolutionPi[2], running Eclipse 4diac runtime and CODESYS soft-PLC simulation, running CODESYS runtime [34]. Additionally, there is an industrial router INSYS icom MRX5[3] that runs a Linux container, which also executes an Eclipse 4diac runtime. The latter device is used to run the orchestrator, which simulates a workstation controller that wraps the underlying devices' low-level atomic skills in composite ones, offering the next layer of an abstraction. The red and green light controls are implemented in IEC 61499 using Eclipse 4diac, whereas the yellow lamp control is implemented in IEC 61131-3 using CODESYS. Each of the atomic skills, which turns the corresponding lamps on and off, is represented in the OPC UA namespace with five methods and state variable as it is shown in Figure 4. However, for these atomic skills there are only two methods realized: *start()*, which turns a lamp on, and *cancel()*, which turns a lamp off. Calling *reset()*, *resume()*, and *suspend()* has no effect in this case. Such modelling is fully compliant with the OPC UA Programs specifications, where all control methods are defined as optional.

The orchestrator can control the execution of all three underlying atomic skills via OPC UA. It also can build sequences of skills, wrap those sequences in composite skills, and expose these skills to the higher levels of control using the same OPC UA skill interface. Here the orchestrator uses the skill state machine shown in Figure 2 as follows:

[1] http://www.wago.us/products/components-for-automation/modular-io-system-ip-20-750753-series/plc/pfc200/

[2] https://revolution.kunbus.com/

[3] https://www.insys-icom.com/en/products/router/mrx-series



[H]

Fig. 3. Pack ML State Machine modeled as an OPC UA Program

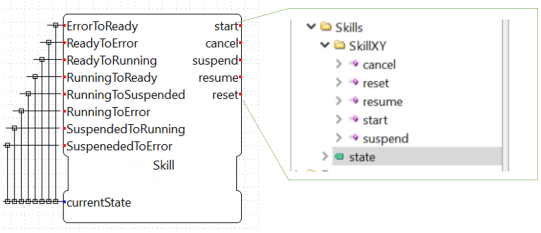


Fig. 4. A Skill Function Block in IEC 61499 and its Corresponding Representation in OPC UA Namespace

- After being initialized, the skill is in Halted state - it turns on the red light via calling an OPC UA *LightRed.start()* method on the corresponding PLC. Additionally, it calls *LightGreen.cancel()* and *LightYellow.cancel()* methods on the PLC's, controlling the green and yellow lights.
- To enter the Ready state, a Client connected to the Orchestrator should call *Orchestrator.reset()* method. The orchestrator, in return, calls *Light.cancel()* methods on all PLC's to turn all the lamps off.
- A Client then can call *Orchestrator.start()* method, which brings the Orchestrator in the Running state, simulating a production. The orchestrator calls *LightGreen.start()*, *LightYellow.cancel()*, and *LightRed.cancel()* on the corresponding OPC UA Servers.
- To pause a running production, a Client calls *Orchestrator.suspend()*. The orchestrator turns the yellow lamp on by calling *YellowLamp.start()* and *LightGreen.cancel()*, *LightRed.cancel()*.

A Client can then use the skill interface to control either each of the low-level atomic skills separately or use the composite skill, running on the orchestrator device, that aggregates the low-level functionality as described above.

Note that example is running on four different devices from four different vendors that communicate about their skills over OPC UA. Moreover, the yellow lamp is controlled by a soft-PLC, which runs CODESYS runtime and its skill is realized in IEC 61131-3 whereas the rest of the application is implemented in IEC 61499. Using an OPC UA interface as a common skill description allows to abstract the low-level functionality implementation and show interoperability between different software realizations running on various devices.

V. Conclusion

The current contribution shows how a skill-based engineering approach can be implemented using the concepts of OPC UA Programs, utilizing its Finite State Machine to describe skill execution logic and observe its behavior in the runtime. Having one common interface for all skill implementations allows the service consumers to easily control each separate skill, as well as the sequences of the skills, available in the system to execute them in a specific order, fulfilling a production task.

This common interface is only possible if the manufacturers agree to use a generic OPC UA interface proposed, which means more development effort. However, our approach promises to increase the overall system effectiveness due to interoperability and re-usability.

As a possible future work, one can mention an implementation of a skill as a pure OPC UA program, as soon as its support appears in the OPC UA stacks.

Acknowledgment

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 680735, project

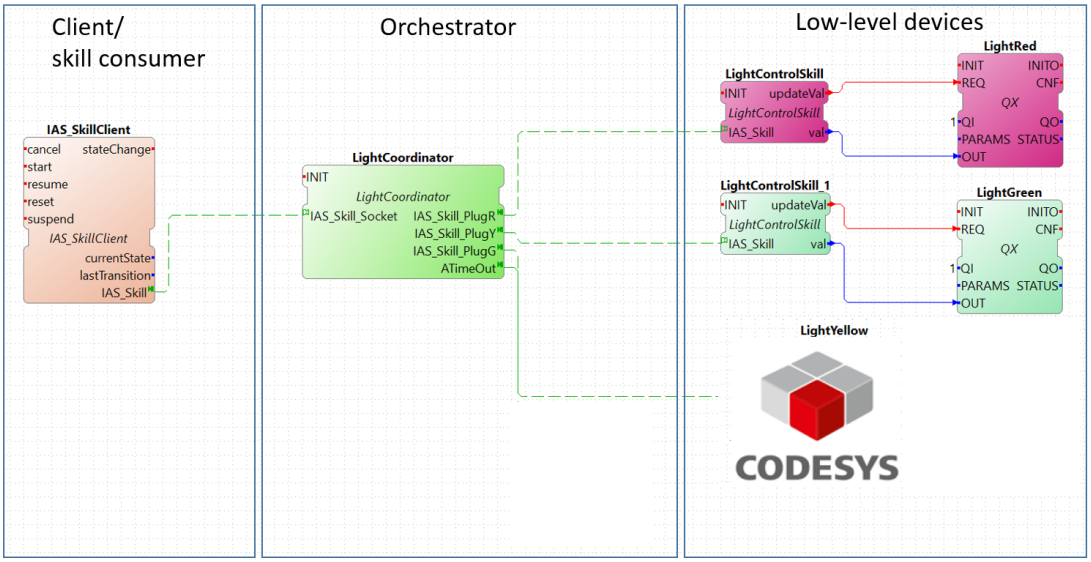


Fig. 5. An Example Application

openMOS (Open Dynamic Manufacturing Operating System for Smart Plug-and-Produce Automation Components).

REFERENCES

[1] Arbeitskreis TB Konzept Normung zu Industrie 4.0 des DKE-Fachbereichs Leittechnik (FB 9), "Die Deutsche Normungs-Roadmap Industrie 4.0. DKE Deutsche Komission Elektrotechnik Elektronik Informaationstechnik im DIN und VDE," Tech. Rep., Dec. 2013.

[2] F. Basile, P. Chiacchio, and D. Gerbasio, "On the implementation of industrial automation systems based on PLC." in *IEEE Transactions on Automation Science and Engineering, Vol. 10, No. 4, pp. 990-1003*, 2013.

[3] K. H. John, and M. Tiegelkamp, "SPS-Programmierung Mit IEC 61131-3," 2009.

[4] J. Delsing, "Local Cloud Internet of Things Automation," Dec. 2017.

[5] "Industrie 4.0 auf dem Weg zu einem Referenzmodell," VDI/VDE-GMA-Fachausschuss, Tech. Rep., 2014.

[6] "Arbeitskreis Industrie 4.0. *Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0*. Forshungsunion im Stifterverband für die Deutsche Wirtschaft e.V." Tech. Rep., 2012.

[7] "Distributed Control Applications: Guidelines, Design Patterns, and Application Examples with the IEC 61499." 2016.

[8] International Electrotechnical Commission, "IEC SC65B. IEC 61131-3 Programmable Controllers Part 3: Programming languages." Tech. Rep., 2003.

[9] I. SC65B, "IEC 61499-1 Function Blocks for Industrial Process Measurement and Control Systems. Part I: Architecture," 2005.

[10] R. Lewis and A. Zoitl, *Modelling Control Systems Using IEC 61499*. The Institution of Engineering and Technology; 2 edition, 2014.

[11] P. Ferreira, and N. Lohse, "Configuration model for evolvable assembly systems," 2012.

[12] K. Dorofeev, C.-H. Cheng, P. Ferreira, M. Guedes, S. Profanter, and A. Zoitl, "Device Adapter Concept towards Enabling Plug&Produce Production Environments," in *Emerging Technologies And Factory Automation (ETFA)*, Limassol, 2017.

[13] N. M. Josuttis, *SOA in Practice*. O'Reilly Media, Inc, 2007.

[14] F. Jammes and H. Smit, "Service-oriented paradigms in industrial automation," Feb. 2005.

[15] Reference Architecture Foundation for Service Oriented Architecture Version 1.0. [Online]. Available: http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/cs01/soa-ra-v1.0-cs01.pdf

[16] SOCRADES - Service-Oriented Cross-layer Infrastructure for Distributed Smart Embedded devices. [Online]. Available: http://socrades.net

[17] H. Bloch, A. Fay, M. Hoernicke, "Analysis of service-oriented architecture approaches suitable for modular process automation," 2016.

[18] H. Bloch, A. Fay, M. Hoernicke, S. Hensel, A. Hahn, L. Urbas, T. Knohl, J. Bernshausen, "A Microservice-Based Architecture Approach for the Automation of Modular Process Plants," 2017.

[19] OPC Foundation, "OPC UA Specification," OPC Foundation, Tech. Rep., 2015. [Online]. Available: https://opcfoundation.org/developer-tools/specifications-unified-architecture

[20] Plcopen and opc foundation: Opc ua information model for iec 61131-3. [Online]. Available: http://plcopen.org/pages/tc4_communication/downloads/plcopen_opcua_information_model_%20v100.pdf

[21] Opc ua with iec 61499. [Online]. Available: https://www.eclipse.org/4diac/documentation/html/communication/opc_ua.html

[22] M. Melik-Merkumians, T. Baier, M. Steinegger, W. Lepuschitz, I. Hegny, and A. Zoitl, "Towards OPC UA as portable SOA middleware between control software and external added value applications," in *Emerging Technologies And Factory Automation (ETFA)*, 2012.

[23] B. Vogel-Heuser, D. Schutz, T. Frank, and C. Legat, "Model-driven engineering of Manufacturing Automation Software Projects A SysML-based approach," 2014.

[24] M. Melik Merkumians, M. Baierling, and G. Schitter, "A Service-Oriented Domain Specific Language Programming Approach for Batch Processes," in *Emerging Technologies And Factory Automation (ETFA)*, Berlin, 2016.

[25] T. Arai, Y. Aiyama, Y. Maeda, M. Sugi, and J. Ota, "Agile assembly system by plug and produce," 2000.

[26] OPC Foundation, "OPC UA Specification Part 10: Programs. Release 1.03," OPC Foundation, Tech. Rep., 2015. [Online]. Available: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-10-programs/

[27] Unified Automation .NET Based OPC UA Client & Server SDK. [Online]. Available: https://www.unified-automation.com/products/server-sdk/net-ua-server-sdk.html

[28] open62541 - an open source C (C99) implementation of OPC UA. [Online]. Available: https://open62541.org

[29] Eclipse Milo. [Online]. Available: https://projects.eclipse.org/projects/iot.milo

[30] Free OPC-UA Library - Open Source C++ and Python OPC-UA Libraries. [Online]. Available: https://github.com/FreeOpcUa

[31] H. Herrero, J. L. Outón, M. Puerto, D. Sallé, and K. López de Ipiña, "Enhanced flexibility and reusability through state machine-based architectures for multisensor intelligent robotics," 2017.

[32] "ISA-TR88.00.02 Machine and Unit States: An Implementation Example of ISA-88," Tech. Rep., 2008.

[33] 4diac - IEC 61499 Implementation for Distributed Devices of the Next Generation. [Online]. Available: https://www.eclipse.org/4diac/

[34] CODESYS - IEC 61131-3 Integrated Development Environment. [Online]. Available: https://www.codesys.com/