Using IEC 61499 and OPC-UA to implement a self-organising plug and produce system

Jörg Walter[1], Kim Grüttner[1], Wolfgang Nebel[2]

[1] OFFIS – Institute for Information Technology, Oldenburg, Germany
[2] University of Oldenburg, Germany
{joerg.walter, kim.gruettner}@offis.de, wolfgang.nebel@uni-oldenburg.de

Abstract In industrial production systems, robotic arms are a common type of universal production facility. Traditionally, they serve a fixed purpose within a production chain. From the Industry 4.0 context, the idea of "Plug and Produce" has emerged, which aims to make production facilities more universally and dynamically usable.
We present a miniature model of an industrial production line that supports Plug & Produce. We programmed all control systems using 4diac, an implementation of IEC/ISO standard 61499, and used OPC-UA as the communication layer. From building this system, we found that the chosen tools are well suited for implementation, although modelling the system would benefit from an abstraction on top of IEC 61499.

1 Introduction

Industrial manufacturing is an important application for robotic systems. Recent trends in production technology lead to customised mass production, down to a lot size of one. These changes, sometimes subsumed under the term 'Industry 4.0', lead to a demand for highly dynamic production facilities. At the same time, manufacturers want to reduce time-to-market and reduce overall cost.

One idea that addresses all three issues is Plug and Produce [2]. In such a system, production relies on universal production facilities (e. g. robotic arms) with standardised capabilities that can be employed in different contexts and with minimal setup time. That way, manufacturers can increase production capacity on demand by quickly adding workstations to a production line. Moreover, multipurpose facilities allow regular rearrangement of production lines, e. g. due to seasonal demand or for a prototype production run. Finally, parts of a production line might continue working while other parts of it undergo maintenance. This ultimately maximises utilisation and thus reduces cost.

We built a miniature model of a production line that implements such a system using modern standards for distributed industrial control systems, IEC 61499 and OPC-UA. We show that they are suitable for modelling a fully decentralised, self-organising production line and explore advantages and limitations.

This work is divided into three main parts: We introduce the main standards we used for our implementation in section 2. In section 3, we show details of our implementation. Finally, section 4 discusses the lessons we learned from our model, including related and future work. A conclusion sums up our findings.

2 Modern industry control

2.1 IEC 61499

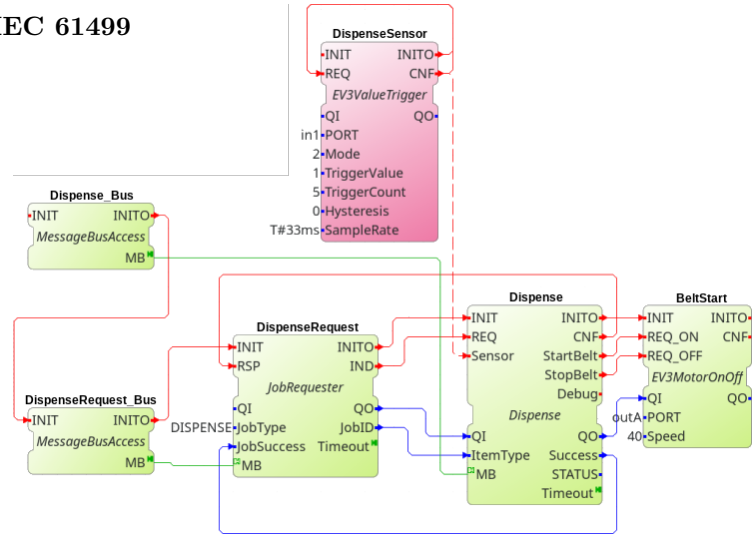


Figure 1. Example of a Function Block Network

IEC 61499 [8] is an international standard that describes an architecture for distributed industry control systems. Among others, it defines a graphical programming language that uses event-based execution of function block (FB) networks (figure 1). A function block can be of three different kinds:

Basic Function Blocks (BFBs) execute an Execution Control Chart (ECC, figure 2), which is similar to a state machine. States trigger code sequences (called *algorithm*s, depicted as yellow boxes) each time they are entered. These can be written in any of the languages defined by the classic industry control standard IEC 61131, e. g. Structured Text.

Composite Function Blocks (CFBs) execute a nested network of function blocks, allowing hierarchical modelling.

Service Interface Function Blocks (SIFBs) execute functionality specified outside of IEC 61499, e. g. native code written in C++.

Function blocks are connected via three different kinds of edges:

Event connections (red in figure 1) trigger execution of the receiving block.

Data connections (blue) transport typed data. They are always associated with one or more events.

Adapters (green) encapsulate event and data connections in both directions and serve as an abstraction similar to object orientation.

IEC 61499 employs an application-centric modelling approach. This means that developers write applications with little regard to its distributed nature. In a separate step, they specify the deployment of applications to execution resources (denoted by FB colour in figure 1) and specify how they communicate.

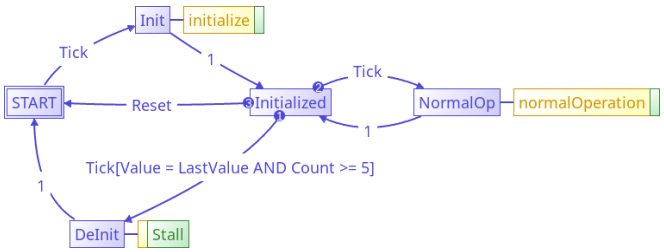


Figure 2. Example of an Execution Control Chart (ECC)

2.2 OPC-UA

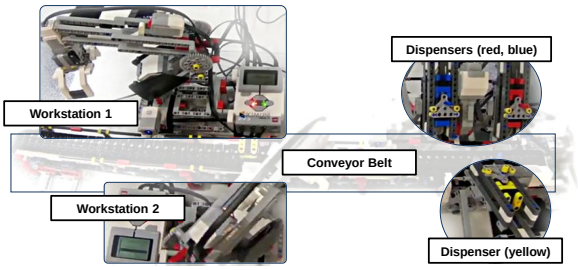
OPC-UA [14] (standardised as IEC 62541) is a vendor-neutral communication protocol intended for industrial automation applications and mainly deployed over TCP/IP. OPC-UA employs a mesh architecture, i. e. every control system can provide its own OPC-UA server, and clients access it directly. A server exposes a tree structure of *nodes* containing *attributes* (typed variables), *methods* (typed function calls), and *events* (e. g. a data changed event for an attribute). Nodes are namespaced, and a globally predefined OPC-UA namespace contains metadata (including type information) about every node. Additionally, there are discovery facilities for different granularity levels (e. g. servers or nodes).

Part 14 of OPC-UA adds a publish/subscribe message bus. With this communication facility, *publishers* provide global messages that all interested *subscribers* receive without the publisher having to know the set of subscribers. This bus can use multi-cast UDP/IP as transport, which means there is no need for a central message broker. When paired with Ethernet Time-Sensitive Networking (TSN), OPC-UA has real-time capability that rivals established field buses [5].

3 Implementation

3.1 Example scenario

(a)



(b)

Figure 3. (a) Physical and (b) logical structure of the miniature model

For our miniature model (see figure 3), we selected the task of commissioning shipments consisting of multiple items that are held in an automated storage facility. We built the mechanical setup using LEGO bricks, while the control systems consist of three LEGO Mindstorms EV3 units and a Raspberry Pi. These are directly supported by 4diac RTE, using Linux as an operating system. Additionally, we used Python for some auxiliary parts. There are four different kinds of control system:

The job scheduler is the top-level unit that manages a database of open commissioning jobs. It watches for available workstations and dispatches jobs to them. It runs as a Python program on the Raspberry Pi and offers a user interface to add and monitor commissioning jobs.

Workstations are robotic arms that know how to take items from a platform right in front of them and to put them into packages at the side of them. Each workstation runs exclusively on a single EV3 unit via 4diac RTE. They are able to shut down safely after finishing their current task (if any).

The transport system is a conveyor belt that moves items from the storage facility to the workstations. It runs on the Raspberry Pi via 4diac RTE.

Dispensers form the automated storage with one dispenser per available item type. They can drop items onto a platform in front of them. Three of them run on one EV3 unit via 4diac RTE.

The job scheduler dispatches a job, which causes dispensers to drop items onto the transport system, which in turn moves the items to a workstation, which then puts them into a package. Each job has associated data that specifies which items are dispensed, transported, and packaged in which order. The job scheduler will only dispatch as many jobs at a time as the production line can process.

Some control systems may be present multiple times, most importantly workstations and storage units. Multiple workstations increase the number of jobs that can be processed in parallel, while multiple dispensers increase the selection of items that can be commissioned. In theory, even multiple job schedulers or multiple or alternate transport systems are possible, but we didn't explore such a configuration. In our experiments, we vary the number of workstations.

3.2 Decentralised control

Traditionally, a central controller would coordinate the actions of all control systems. For our goal, this would mean that it would have to detect how many and what kind of stations are present. We chose a self-organising approach based on a service-oriented architecture instead, which leads to simpler, more encapsulated programming and less tight dependencies between components. The setup does not use a dedicated service coordinator; stations use the message bus to negotiate services among themselves.

Each control system (CS) offers any amount of services to other CSes (*provider*). It may in turn request services from others as required (*consumer*). Figure 3 shows the chain of services offered and requested:

1. The job scheduler requests the COMMISSIONING service. If no one is able to perform this service, it would try again after a workstation signals readiness.
2. A workstation accepts the job and becomes responsible for its execution. It uses the TRANSPORT service to fetch items until the job is complete.
3. The transport system requests DISPENSING of the correct item, and starts and stops the belt at appropriate moments. Figure 1 shows a simplified version; the *Dispense* FB contains an ECC that manages this functionality.
4. A dispenser in the automated storage facility dispenses one item.

This ends the chain of requests. Responses will be sent in reverse order, with each CS performing its own task in between. Finally, the workstation will signal completion of the job to the job scheduler, which will then issue the next job.

3.3 Communication

For requests and responses, our setup uses an OPC-UA based Publish/Subscribe message bus. We defined four protocols that consist of a sequence of messages sent over the bus:

Service Negotiation The protocol consists of five messages, sent in this order: AVAILABLE (Consumer→all Providers), REQUEST (P→C), ASSIGN (C→P), ACKNOWLEDGE (P→C), COMPLETE (P→C)
AVAILABLE and REQUEST may be ignored temporarily, e. g. while busy. On the other hand, REQUEST may be sent without a prior AVAILABLE message. ASSIGN starts the mandatory part of the protocol. These messages all contain a unique identifier for their task. If ACKNOWLEDGE or COMPLETE do not occur after a sufficient timeout, the consumer may assume provider failure. COMPLETE includes a success indicator or error code.

Task Queue In order to decouple services further, we inserted task queues into the service protocol: The actual consumer submits a task to the task queue, whose presence is implied. The interface are two messages, ADD and COMPLETED. The task queue subsequently performs the Consumer side of the Service Negotiation protocol.

Task Data This protocol transfers commissioning details from the consumer to the provider. It consists of messages to retrieve the number of items to commission, and each item's type.

Conveyor Control This protocol serves two purposes: The robotic arms need time to grab items, and it must know the contents of the conveyor belt so that a sensor barrier is sufficient to find out which item to grab.
For this reason, every action that changes the belt's contents is announced on the message bus, so that all parties can update their locally-held view of the belt's contents. This also allows dynamic changes of stations, as no station makes assumptions about other stations' existence or position.

The first two protocols map to Execution Control Charts (ECCs) in a straightforward manner, with the task ID stored in an ECC shared variable. The conveyor control protocol needs more complex processing, as most messages lead to an

update of the local view of the belt's contents. We wrote and tested an ECC *algorithm* that performs these updates, and every state executes that algorithm on entry. That way, ECCs combine the benefit of code reuse with the graphical modelling of a state machine.

3.4 Self-organised scheduling

As it has been described up to now, the job scheduler only schedules the top-level Commission tasks, everything else is implied by the Service Negotiation protocol, i. e. mainly first-come-first-serve. However, the job scheduler also implements all required task queues. These are effectively schedulers for their associated service. In our setup they were simple FIFO queues, but there is no technical requirement for that. Using other algorithms, we could have regained some control over job execution order, e. g. prioritising one job with all its individual tasks over all others, while still having a fully dynamic production facility.

3.5 Manual dispensing

In order to show how services may be implemented in completely different ways, we added a Python-based provider for the Dispense service. It watches for Dispense requests that are not answered within a certain time and displays a message on the user interface, so that a human operator can instead fulfil this request. This demonstrates the extensibility such a dynamic architecture provides.

3.6 Python-based message broker

We use OPC-UA as the decoupling layer between services. Unfortunately, the native OPC-UA PubSub message bus has only officially been released in 2018 [15], so the latest release of 4diac at the time of implementation (1.8.4) did not support it yet. As a quick workaround, we implemented a message broker in Python using OPC-UA attributes, methods, and events. It mimicks a subset of what the native message bus would have offered.

On the IEC 61499 side, we encapsulated this in a CFB and used an Adapter (see *MessageBusAccess* in figure 1) to connect each message-bus-using function block to the bus. This way, we can easily switch to native PubSub messaging once it becomes available in an upcoming release of 4diac.

4 Discussion & future work

The factory model performed its job correctly for hours, which we demonstrated at a public trade fair: While a batch of jobs was executing, we regularly shut down a workstation, placed it at a different (logical and physical) location on the conveyor belt, and booted it up again. The remaining workstation was not affected by this, and the re-added workstation automatically started performing jobs after boot-up. As a second scenario, we removed both workstations, halting

production. After adding them back, all jobs were completed correctly. The speed of the conveyor belt was the main limitation for job throughput, with movement of the robotic arm as a secondary factor; latency due to service negotiation was completely hidden by mechanical processes.

We did have occasional errors. Main sources were mechanical weaknesses of the chosen miniature modelling toolkit and false positive readings of its sensor barriers. In rare circumstances, computational overhead of the message bus emulation led to surprising execution orders, but this did not impact the (ultimately correct) outcome. We identified shortcomings that should be addressed in future work; they concern timing, functionality, and modelling aspects.

4.1 Timing properties

In a cyber-physical system, timing accuracy is often a prerequisite for functional correctness. Our setup performed fast enough for uninterrupted and correct operation, but there is no formal guarantee that this will always be the case.

IEC 61499 and OPC-UA lack a formal methodology to ensure a hard upper bound to end-to-end or single-service latencies. While existing worst-case timing analysis is probably able to ensure a single service's timing, the self-organising and dynamic nature of our setup poses additional challenges. Likewise, we also cannot guarantee the overall throughput.

These difficulties are a consequence of the fact that we do not tightly synchronise control units. Any approach using loose coupling would have similar issues; since these are well researched, future work should explore known solutions and their applicability to IEC 61499/OPC-UA.

4.2 Functional correctness

Self-organising plug and produce worked well using a service-oriented architecture over a publish/subscribe message bus (even when emulated). The system stayed functionally correct at all times: We could add and remove workstations in the middle of a production run. New stations automatically picked up jobs after booting up, and after a station was shut down, remaining stations completed all jobs. Shutting down a station and re-adding it during the same production run at a different location worked flawlessly as well.

IEC 61499 semantics are incompletely specified [20], which limits interoperability between different implementations. Since we used 4diac as our particular run-time environment (RTE), this was not a notable problem; we simply relied on the semantics it implements. Loose coupling via services was helping here, because the service abstraction allowed us even to use different languages for different services and would have allowed different RTEs as well.

4.3 Modelling aspects

Modelling in IEC 61499 was quite productive most of the time; its modularisation facilities met the expectations we would have had on other programming

languages. The application-centric view greatly improved separation of concerns while modelling, especially when mapping multiple independent functionalities to the same control system.

We used Execution Control Charts extensively, as their practical applicability proved to be high. Unfortunately, introducing error handling and return into a known safe state often doubled ECC size. Hierarchical states would have been useful for our use case and would have made ECCs more maintainable and robust.

In the end, IEC 61499 worked well for modelling strongly coupled applications running across multiple devices. For loosely coupled systems of such applications, we lacked a more abstract way of modelling and analysing their interdependencies. We could not answer questions like: 'Does adding or removing a particular station allow the system to perform all of its scheduled jobs?' or, when taking time into account: 'Does adding another workstation actually improve throughput?' Functional correctness at this level was only an observation, not a property we systematically ensured.

4.4 Related work

There are various works that describe Plug and Produce (P&P) or similar manufacturing systems. The concept of Holonic Manufacturing Systems [18] is basically what we implemented, just with modern industry standards. Our technique could be used to build systems using the PROSA reference architecture [6].

Many P&P approaches like [3] or [16] rely on a central configuration instance instead of decentralised self-organisation like we do. This improves analysability and scheduling over our approach; while we didn't explore that option, our task queues do allow more controlled scheduling, even in a distributed manner.

The authors of [1] show an agent-based approach to P&P using a very fine service granularity. We target a higher level of service abstraction to exploit the advantages of both, tightly coupled encapsulated control systems and their dynamic collaboration.

Within the IEC 61499 community, P&P-like systems have been implemented early on [4], and recently using 4diac [10]. These are still centrally controlled, however, and facilitate adaption through dynamic reprogramming of control systems. The approach in [13] has a higher degree of self-organisation, much closer to our approach. But like before, they model high-level processes in IEC 61499, while we decouple through OPC-UA; this eliminates the need for on-the-fly reprogramming and a homogeneous software stack but introduces modelling challenges as shown above.

4.5 Future work

For functional and extra-functional correctness of IEC 61499, several approaches exist: Some approaches use a more rigidly specified subset of IEC 61499 in order to be able to perform formal analysis [21], while others use contract theory to monitor and ensure properties [17].

Unfortunately, cyber-physical systems always have to account for physical failures that cannot be exhaustively modelled in a formal framework, if only for the reason that there is no way to enumerate all possible failure modes. Thus, error detection and handling is inevitable, and a monitoring strategy based on contracts can combine formal analysis with real-world constraints.

The most important aspect for dynamically cooperating robotic systems is the modelling abstraction. While IEC 61499 allows a model-based approach, it lacks an abstract view on services and protocols and dynamic changes in them.

An obvious candidate for abstract modelling is UML and its derivative SysML. This is especially promising as it is already being used for contract-based design in other fields of application [7,11].

Agent-based modelling like the PROSA reference architecture [6] addresses allocation and scheduling aspects by encapsulating coordination tasks in agents, but it still lacks an a priori modelling and validation approach.

The Arrowhead Framework [19] provides methods to model and manage large collaborative automation systems. It has significant overlap with OPC-UA and it is unclear how well these two cooperate. Moreover, Arrowhead still has to address validation challenges.

The authors of [12] present a formal assessment methodology to compare service configurations against each other. This approach might be useful, although it assumes a more centrally controlled setup.

Finally, dynamic contracts [9] try to bridge the gap between contract-based design and dynamically changing systems.

So far, none of these approaches provide a unified solution for a model-driven workflow with the ability to ensure functional correctness and timing constraints on industry control systems. An obvious way forward would be to test combinations of these tools and methods in order to address the weaknesses we identified.

5 Conclusion

We have demonstrated a working, fully self-organising model of a dynamically cooperating production system, robotic arms in this particular case. With IEC 61499 and OPC-UA we used two notable modern industry standards that are well suited to building systems of this complexity.

Our experience was mostly positive, but some open issues remain. Semantic issues and timing analysis of IEC 61499 remain problematic. One important insight is that we would profit from another layer of abstraction on top of IEC 61499 in order to model systems with more dynamic cooperation and less central control than currently in use.

We have also investigated some methodologies that address the identified challenges. In future work, we need to explore how we can integrate IEC 61499 with those. Among these options, contract-based design methodologies and service modelling frameworks seem to be the most promising complements.

Acknowledgement This work has been funded by the Ministry of Economic Affairs, Employment, Transport, and Digitalisation and the Ministry of Science and Culture of the federal state of Lower Saxony, Germany.

References

1. Antzoulatos, N., Castro, E., Scrimieri, D., Ratchev, S.: A multi-agent architecture for plug and produce on an industrial assembly platform. Production Engineering **8**(6), 773–781 (Dec 2014)
2. Arai, T., Aiyama, Y., Maeda, Y., Sugi, M., et al.: Agile Assembly System by "Plug and Produce". CIRP Annals **49**(1), 1–4 (2000)
3. Arai, T., Aiyama, Y., Sugi, M., Ota, J.: Holonic assembly system with Plug and Produce **46**, 289–299 (10 2001)
4. Brennan, R.W., Fletcher, M., Norrie, D.H.: An agent-based approach to reconfiguration of real-time distributed control systems. IEEE Transactions on Robotics and Automation **18**(4), 444–451 (Aug 2002)
5. Bruckner, D., Blair, R., Stanica, M.P., Ademaj, A., et al.: OPC UA TSN. Industrial Ethernet Book (105/9) (Apr 2018)
6. Brussel, H.V., Wyns, J., Valckenaers, P., Bongaerts, L., et al.: Reference architecture for holonic manufacturing systems: PROSA. Computers in Industry **37**(3), 255–274 (1998)
7. Ehmen, G., Grüttner, K., Koopmann, B., Poppen, F., et al.: Coherent Treatment of Time in the Development of ADAS/AD Systems: Design Approach and Demonstration. In: SAE Technical Paper. SAE International (Apr 2018)
8. International Electrotechnical Commission: IEC 61499-1: Function blocks – Part 1: Architecture (Nov 2012)
9. Kim, E.S., Sadraddini, S., Belta, C., Arcak, M., Seshia, S.A.: Dynamic contracts for distributed temporal logic control of traffic networks. In: IEEE 56th Annual Conference on Decision and Control (CDC). pp. 3640–3645 (Dec 2017)
10. Lepuschitz, W., Zoitl, A., Vallée, M., Merdan, M.: Toward Self-Reconfiguration of Manufacturing Systems Using Automation Agents. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) **41**(1), 52–69 (Jan 2011)
11. Mazzini, S., Favaro, J.M., Puri, S., Baracchi, L.: CHESS: an Open Source Methodology and Toolset for the Development of Critical Systems. In: Joint Proceedings of EduSymp 2016 and OSS4MDE 2016. pp. 59–66 (2016)
12. Neves, P.: Reconfiguration Methodology to improve the agility and sustainability of Plug and Produce Systems. Ph.D. thesis, KTH Royal Inst. of Technology (2016)
13. Onori, M., Lohse, N., Barata, J., Hanisch, C.: The IDEAS project: Plug & produce at shop-floor level **32**, 124–134 (04 2012)
14. OPC Foundation: OPC Unified Architecture – Overview and Concepts (Nov 2017)
15. OPC Foundation: OPC Unified Architecture – Part 14: PubSub (Feb 2018)
16. Pfrommer, J., Stogl, D., Aleksandrov, K., Navarro, S.E., et al.: Plug & produce by modelling skills and service-oriented orchestration of reconfigurable manufacturing systems. at-Automatisierungstechnik **63**(10), 790–800 (2015)
17. Tran, D.D., Walter, J., Grüttner, K., Oppenheimer, F.: Towards Contract Based Assertions in IEC 61499 Applications. In: FDL 2018 (2018)
18. Valckenaers, P., Van Brussel, H., Bongaerts, L., Wyns, J.: Holonic manufacturing systems. Integrated Computer-Aided Engineering **4**(3), 191–201 (1997)
19. Varga, P., Blomstedt, F., Ferreira, L.L., Eliasson, J., et al.: Making system of systems interoperable–The core components of the arrowhead framework. Journal of Network and Computer Applications **81**, 85–95 (2017)
20. Vyatkin, V.: The IEC 61499 standard and its semantics. IEEE Industrial Electronics Magazine **3**(4), 40–48 (Dec 2009)
21. Yoong, L.: Modelling and Synthesis of Safety-critical Software with IEC 61499. Ph.D. thesis, ResearchSpace Auckland (2010)