Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 118 (2023) 56–61

www.elsevier.com/locate/procedia

16th CIRP Conference on Intelligent Computation in Manufacturing Engineering, CIRP ICME ‘22, Italy

Functionality test methodology for virtual commissioning of reconfigurable manufacturing systems

Bernhard Wallner[a,*], Thomas Trautner[a], Friedrich Bleicher[a]

[a] *TU Wien, Institute of Production Engineering and Photonic Technologies, Getreidemarkt 9, 1060 Wien, Austria*

* Corresponding author. Tel.: +43-1-58801-31120; *E-mail address:* wallner@ift.at

Abstract

Reconfigurable manufacturing systems (RMS) are complex systems that regularly change the system behavior. One method to deal with this is virtual commissioning (VC). The new behavior needs to be implemented in the individual machines and the RMS-control and tested and verified afterward. Although various concepts for modeling behavior exist, simulations often lack interpreting and testing. This paper proposes a methodology for simulating and testing the functionality of RMS based on different behavior descriptions. A Proof of concept shows the first implementation of this methodology using a CAE platform combined with python.

Keywords: RMS; Test methodology; Virtual commissioning

1. Introduction

Reconfigurable Manufacturing Systems (RMS) are modular and complex and can be adapted regularly to meet changed requirements. These changes include switching hardware components like machine tools, industrial robots, autonomous mobile robots (AMR), sensors, measuring devices, and software components representing the hardware like controllers, interfaces, or parameter sets.

The digitization trend of the last years complicates these changes due to the rising number of internet of things (IoT) devices used in manufacturing. Additional or changed devices often come with changes in the provided interfaces, making the integration into the cell control difficult.

Programming environments are filled with enormous functionality, making control software more complex and extensive. Changes in cell control still require human interaction and programming skills and are therefore afflicted by human errors. The cell control program must be tested and verified to eliminate these errors, which becomes even more challenging in complex systems like RMS.

Virtual commissioning (VC) is often used in these scenarios and should be capable of reducing implementation times and helping to eliminate errors. These solutions include proprietary simulations and are entangled within the software.

This paper focuses on simple manufacturing cells for machine tending, typically consisting of a machine tool, an industrial handling robot, and some peripherical devices like vision systems. Findings also may be applicable for upscaling to complete RMS.

We begin with a brief overview of state of the art regarding simulation and VC, control technologies, and cell control architectures. Subsequently, we describe the proposed methodology for functional testing in chapter 3. To better understand, we explain the methodology using a proof of concept. In chapters 5 and 6, we present our findings, evaluate the methodology’s potential, and conclude. Also, a brief outlook for further research activities is given.

10.1016/j.procir.2023.06.011

2. State of the art

2.1. Simulation methods and Virtual Commissioning

Simulation methods are widely used to test specific functions within a manufacturing system. Especially the planning of robotic system applications is standard with those systems. Often the functionality focuses on the primary operation of the machine in focus (e.g., path planning for robot controllers) but lacks side functions (e.g., integration of external devices' control logic).

Also, VC is performed in closed systems and is highly entangled with the control software or logic. This relation results in some disadvantages for these methods: (1) due to the often highly proprietary solutions, portability of solutions is low, especially for machines from different manufacturers, and (2) due to the restricted simulation possibilities, it is not possible to test the whole control logic at once.

The drawbacks are especially true for PLC simulation since simulation environments are provided mainly by the PLC providers themselves.

VC will become more important in future years for research in optimizing data analysis – especially with the usage of machine learning. For example, Zhang et al. [1] show a VC attempt to test machine learning algorithms in a Siemens NX MCD environment for reconfigurable assembly systems.

Additionally, VC-software tools often can be used as a digital model of the physical world. Depending on the implementation and automatic data flow, these digital representations can be considered digital shadows (unidirectional) or digital twins (bidirectional) [2].

2.2. Control technologies

At the different levels of the automation pyramid, various methods are established to model processes or behavior and use these to control arbitrary devices. There is a wide range of aspects and detail levels, from top-level enterprise process models like the Business Process Model and Notation (BPMN) to bottom-level control programming like IEC 61131. As these contain encoded knowledge about the behavior of a system and its components within the scope of the application, we refer to them as behavior descriptions.

All have in common that they require domain experts that understand the processes and machines and have to model and implement the logic into the systems. Modeling and implementation are often done using underlying proprietary software. The Reference Architecture Model Industry 4.0 [3] and cyber-physical systems claim the dissolution of the automation pyramid [4], the borders between the layers blur, and description formats can not be allocated to specific layers anymore.

Since the introduction of the IEC 61131 in the 1980ies, it has been widely used in the manufacturing domain for programming PLCs. The main advantages are predefined functions and low-level programming languages like instruction list, structured text, ladder diagrams, function block diagram, and sequential function chart (SFC) [5]. PLCs often do not support the portability or exchange of programs. AutomationML, PlantUML, PLCOpen, and other standards try to ease the data exchange and counteract the issues caused by proprietary solutions. These three standards are all somehow based on or include IEC 61131.

Although IEC 61499 claims to meet today's challenges like portability, interoperability, reusability, or distribution, the industry still has not accepted it [6]. It is handy for distributed systems because it separates the application from so-called resources where the application subtasks are executed. Like IEC 61131, IEC 61499 also uses function blocks for application modeling. The main difference is an additional event head responsible for managing and triggering the execution of the functionality. To facilitate the transition from IEC 61131 to IEC 61499, the latter also defines and implements IEC 61131 function blocks [7].

UML is often used to model various aspects of the software of interest in software development. The UML standard defines the state charts according to the work of David Harel [8] of the 1980ies, where he combined and extended Mealy- and Moore-machines [9]. Also, State Chart XML (SCXML), an event-based state machine language, is based on Harel State Tables [10].

OPC UA as a platform-independent communication standard has been established over the last years. It allows a semantic interpretation of the machines in focus, which is beneficial for RMS. State machines have been added in part 16 of the standard. The implementation of the state machines focuses on basic functionalities like states, transitions, and substates but does not support complex functions like parallel states [11]. Parallel states may be implemented in future versions of OPC UA as it happened with guards for state machines as proposed by Frühwirth et al. [12].

As the name indicates, BPMN provides a notation for business processes. Business users, integrators, or developers gain understanding due to the graphical aspect [13]. The flow-based processes rely on a traceable token that passes through the nodes. The graphical elements are organized into five categories: flow objects, data, connecting objects, swimlanes, and artifacts.

2.3. Control architectures

For flexible manufacturing systems, some kind of manufacturing management software often is used that supervises the system on the SCADA level of the automation pyramid but is not responsible for control execution.

Within manufacturing cells, there are different possibilities for implementing cell logic. This depends on interfaces provided by the equipment manufacturers, which are therefore often proprietary.

There are some attempts to unify the interfaces, e.g., VDMA 34180 [14], MTConnect, and OPC UA for Machine Tools, but equipment manufacturers often do not or do not implement these standards to their full extent. One possible solution is switching to service-based interfaces, as proposed by Wallner et al. [15]. Another attempt is using a BPMN-based approach for manufacturing orchestration like *centurio.work* as proposed by Pauker et al. [16].

3. Proposed Methodology

Based on the identified demands of chapter 2, we propose a test methodology as shown in Fig. 1.

The central concept of this methodology is modularity which reduces proprietary restrictions and enables the swapping of single modules.

The methodology uses three primary artifacts as input: (1) Behavior Descriptions, (2) Test Cases, and (3) Failure Descriptions, which are marked by the checklist icons in Fig 1. The testing unit processes the test cases and communicates with a predefined simulation using an I/O interface. The output of the test methodology is a test log file that includes all performed test cases with the pre-testing states, completed transitions, and detected failures on which the tested behavior description can be improved.

This methodology makes it possible to separate the behavior description and cell control logic from the simulation or physical system that executes or interprets the control commands. Therefore, different cell controls can be tested with minimum effort for the same simulation setup.

3.1. Behavior Description

Chapter 2.2 described different possibilities for control technologies that represent the proprietary logic.

There are two possibilities to convert the proprietary logic into an exchangeable behavior description: The first is an automatic attempt requiring an abstraction layer to convert the logic into a defined and interpretable format like SCXML, that abstractly describes the available system states, the offered triggers to change those states, and the relevant conditions to make a state transition. The second attempt relies on the domain expert who implemented the proprietary logic and manually creates the behavior description within the documentation process. This may be necessary, as e.g., low-level PLC programming, omits some of the information necessary to define a composed state, and possible transitions.

3.2. Simulation with I/O-Systems

A core feature for proper testing is an appropriate system representation or so-called digital model. The simulation should adequately replace the physical system and therefore (1) provide an I/O interface that is identical in function and labeling and (2) allow detection of the failures of interest.

The I/O interface (1) enables the cell control to use the same logic as it would use to interact with the physical system. Also, it makes hardware in the loop, model in the loop, and software in the loop applications possible

One may also use this digital model and automatic data transition to test digital shadow or digital twin applications. The failure detection (2) enables the functionality of the testing unit described in chapter 3.3.

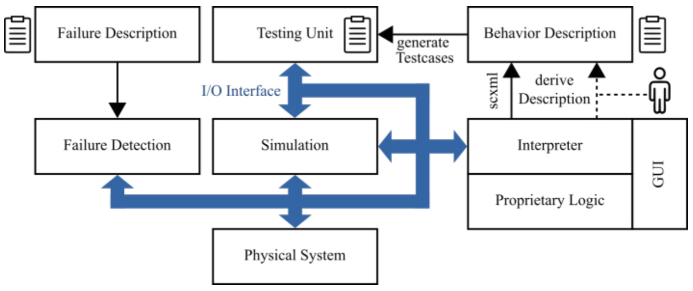


Fig. 1. Proposed Methodology.

3.3. Failure Detection and Test Cases

Failures can occur on all levels of the automation pyramid. Since we want to analyze the behavior description, there are two main failure types we want to discuss in this paper:

Type 1 occurs with a formally correct but faulty behavior description. It includes all kinds of crashes or faults caused by allowed transitions that trigger machine actions or set signals.

Type 2 results from a wrong behavior description of the physical system, including misalignments between targeted states and reached states (wrong or no transition) or deadlocks.

To analyze the behavior description properly, these kinds of failures (1) have to be identified by a domain expert, (2) have to be implemented into and detectable by the simulation or the interpreter, and (3) have to be documented. (1) can be achieved via a widely used method: Failure Mode and effects analysis (FMEA). The advantage of this method is the relation between failures and possible causes, but it can be hard to test within Cyber-Physical Production Systems (CPPS) [17].

Since fault detection within the simulation environment is a discipline of its own, it will not be discussed in detail. How e.g., collision detection can be implemented into a simulation environment is shown by Mei and Lee [18]. For the proposed methodology, it is essential that failures identified by domain experts can also be modeled and detected by the simulation. These failures must be communicated to the testing unit, evaluated, and documented.

Documentation of failures, especially why and when they have occurred, is an essential step for system integrators. Unit tests and test cases are standard in software development to meet this problem. Programming languages often offer specialized libraries for these tasks that cover all necessary steps. E.g., python offers the *unittest* framework that can be used for testing.

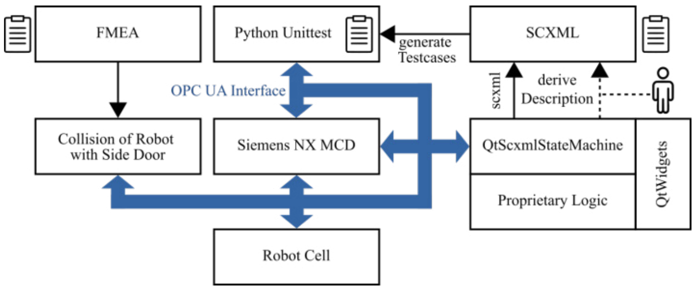


Fig. 2. System Architecture Proof of Concept.

We suggest that the test cases can automatically be derived from the behavior description since the description format is interpretable. Da Silva et al. [19] propose a test specification language (TSL) to describe machine- and human-readable test cases, which might apply to this step. Also, some state chart IDEs like YAKINDU define unit testing and state chart testing languages like SCTUnit.

Each transition must be tested to avoid Type 1 and Type 2 failures. Especially changes that are related to user inputs should be checked. Therefore, the minimum number of test cases corresponds to the available transitions. The number of test cases increases even more if the transitions rely on multiple variables because all variable combinations create additional test cases.

3.4. Adaption of State Machines

After identifying the faulty transitions, domain experts have to adapt the state machine accordingly. The combination of test cases (implying the states and transitions) and failures as the output of the testing unit are beneficial for this task. Deploying the new state machine triggers the next iteration cycle, which can be repeated until no failures are found.

The adaption cycle might be helpful for VC of reconfigurable digital twins where the software can be tested in advance.

Especially for IEC 61499, there exist some promising fault handling attempts: e.g., Leitao et al. [20] propose a reconfiguration architecture for fault handling (RAFAH) that detects fault states and dynamically reconfigures the control system.

4. Proof of concept

We provide a proof of concept including a flexible manufacturing cell implemented in Siemens NX MCD and a Python Program. The system architecture is shown in Fig. 2.

In this use case, we want to investigate the state machine of the cell controller regarding the communication between the machine tool and robot. In particular, we are interested in crashes due to wrong or incomplete logic conditions in the state machine (especially transitions), also known as guards. Two main crashes have been identified via the FMEA: (1) the robot crashes into the closed side door of the machine tool, and (2) the closing side door crashes into the robot which is still inside the machine tool.

Fig. 3 shows the SCXML-file representing a control logic that prevents these crashes. It consists of three parallel sub-state machines for the side door, light curtain, and robot. To test the methodology, we manipulated the state machine to force collisions. An easy way is to remove the guard conditions *In(sd.opened)* for the robot or malfunction the light curtain to stay in the *lc.untriggered* state and not detect the robot occupying the side door.

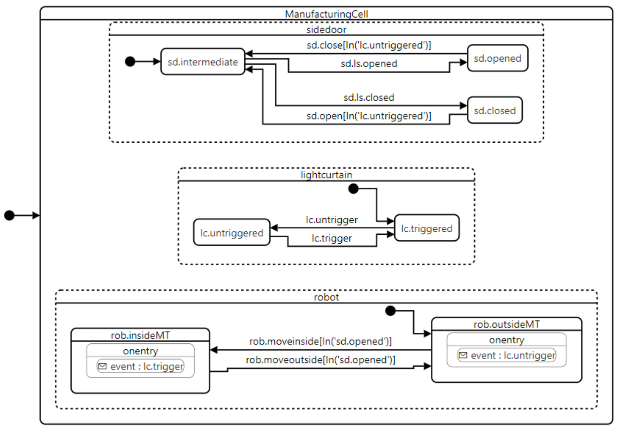


Fig. 3. State machine represented in the SCXML-file.

4.1. Simulation in Siemens NX MCD

Siemens NX MCD is a CAE tool for physics-based simulation of mechanical components. Siemens NX MCD was used because it offers OPC UA functionality which can be used as the I/O interface as proposed by the methodology.

We set up a minimalistic digital representation of a manufacturing cell, consisting of an EMCO 5-axis machine tool and an ABB industrial robot, as shown in Fig. 4. The main functionalities for testing the automation use case were implemented on a sensor/actor level. E.g., the side door for automatic loading is represented by two limit switches indicating the state of the door and a pneumatic piston with a 5/2 directional valve to change the conditions of the door. Also, some robot paths into and out of the machine tool were defined. Lastly, a light curtain guarding the side door was set up to determine if the robot is penetrating the machine tool or not. Fig. 5 shows the robot outside the machine tool (a) and reaching inside (b). The pink triangles represent the intersection between the robot and the light curtain and indicate the trigger of the corresponding signal.

These low-level signals are available via an OPC UA server to monitor and control the simulation during runtime. A complete list of the inputs and outputs of the simulation is described in Table 1.

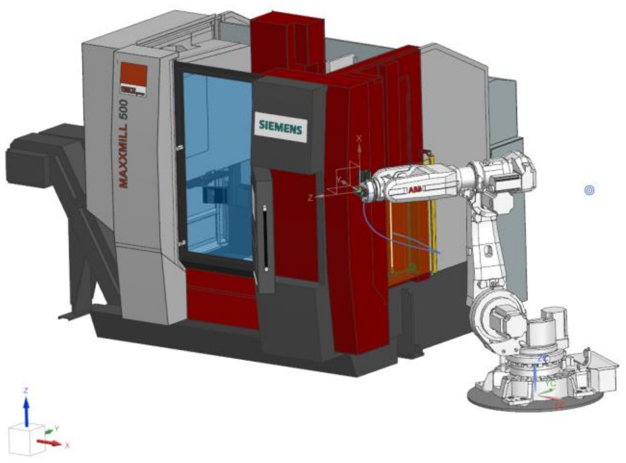


Fig. 4. Proof of concept setup in Siemens NX MCD.

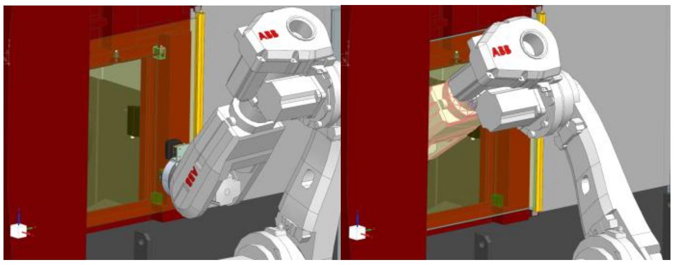


Fig. 5. (a) Robot outside; (b) Robot inside, triggering light curtain.

Table 1. I/O Variables of the Simulation available via OPC UA.

Signal name	Type	Description
SideDoorOpened	Output	Limit switch for the opened side door
SideDoorClosed	Output	Limit switch for the closed side door
SideDoorCrashed	Output	Indicates crash of robot and side door
LightcurtainOccupied	Output	Represents the light curtain guarding the side door
OpenSideDoor	Input	Triggers opening of the side door
CloseSideDoor	Input	Triggers closing of the side door
StartRobotPath	Input	Starts one of three possible robot paths

4.2. Python Implementation

Python was used as the implementation language for the state machine interpreter, GUI, OPC UA interface, and logging functionality. Python is widely used in the scientific community and offers all necessary frameworks and packages to implement the functionality. Table 2 shows a list of the used frameworks with the corresponding versions.

Table 2. Used Python Frameworks.

Framework name	Version
asyncua	0.9.92
python	3.9.7
NumPy	1.21.2
PySide2	5.15.2

The python program performs several tasks asynchronously: First, the program builds up an OPC UA subscription to the signals provided by the simulation using the *asyncua* library. Also, the output signals are set accordingly to the state machine. Second, it imports the SCXML-File, and the state machine is executed in the *QtSCXMLStateMachine*-interpreter. Third, it hosts a GUI to enable user input via *QtWidgets*.

Since the signal labels likely differ between the simulation and the SCXML-File, mapping has been implemented. This also allows configuration for different combinations of simulations and description files.

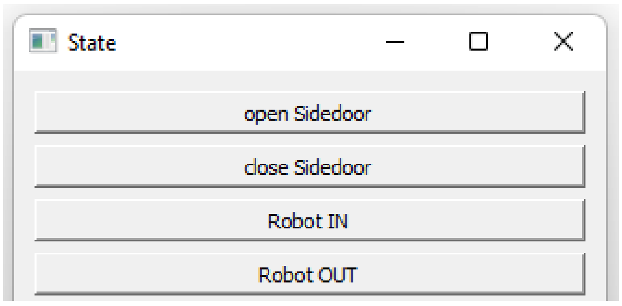


Fig. 6. GUI of the Python Program.

4.3. Test strategy

Since the use case is relatively simple, the tests were performed manually via the GUI of the python program, which is shown in Fig. 6. It allows simple user commands to request opening or closing the side door and for the robot to move in or out of the machine.

Occurring faults were detected and logged automatically. The logs consist of failure type, pre-failure state, user input, and performed transition that led to the failure. Fig. 7 shows a log that documents a crash of the industrial robot into the closed side door.

The *scxml.statemachine-State*-flags indicate the pre-crash states. The *New data change event*-flags show exchanged data via OPC UA - the *Request* indicator documents user inputs via the GUI.

This information eases the identification and debugging of the cell control logic. In this example, the domain expert can debug the transition of the industrial robot penetrating the side door.

5. Findings

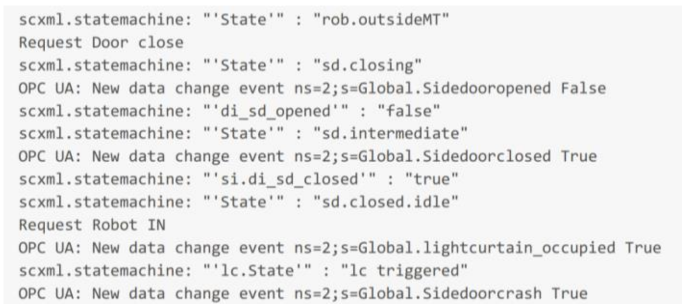
After the first implementation and testing of the methodology, some findings have been identified.

It could be shown that testing the behavior description, especially the failure identification and logging is possible. SCXML can also represent complex state machines and relevant transitions and signals for manufacturing cells. It was also possible to implement the methodology through multiple software tools and frameworks (Siemens NX MCD, Python).

One identified challenge relates to signal mapping. The first implementation still needed manual mapping. It was not easy to generically import the relevant OPC UA nodes into the SCXML-file through the python state machine interpreter. Here we have a disparity between the I/O interface, the behavior description format, and the interpreter that has to be solved in the future to make the exchange easier.

Since the methodology wants to meet the modularity aspect of RMS, it faces similar problems regarding the standardization of the I/O interfaces.

The interpretation was implemented and tested only for one specific behavior description format. Although SCXML fulfilled all requirements and showed promising results, the methodology also has to be validated for other description formats and corresponding interpreters.



```
scxml.statemachine: "'State'" : "rob.outsideMT"
Request Door close
scxml.statemachine: "'State'" : "sd.closing"
OPC UA: New data change event ns=2;s=Global.Sidedooropened False
scxml.statemachine: "'di_sd_opened'" : "false"
scxml.statemachine: "'State'" : "sd.intermediate"
OPC UA: New data change event ns=2;s=Global.Sidedoorclosed True
scxml.statemachine: "'si.di_sd_closed'" : "true"
scxml.statemachine: "'State'" : "sd.closed.idle"
Request Robot IN
OPC UA: New data change event ns=2;s=Global.lightcurtain_occupied True
scxml.statemachine: "'lc.State'" : "lc triggered"
OPC UA: New data change event ns=2;s=Global.Sidedoorcrash True
```

Fig. 7. Excerpt of the crash-log.

One crucial assumption also requires further research: We assumed an automatic or semi-automatic transition between the proprietary logic and the behavior description format. In order to automate this step, it has to be investigated how control technologies like IEC 61131 or IEC 61499 can be transferred to standardized behavior description formats and if additional information or manual tasks are required. This is also applicable to the automatic generation of test cases. Some unification and standardization should be considered.

6. Conclusion and Outlook

This work proposes a methodology to make functionality testing of RMS easier and straightforward. The central aspect lies in modularity and the exchangeability of test modules and automatable steps. Therefore, it eases VC tasks and enables the implementation of digital shadows or digital twins.

The proof of concept showed the first implementation of the methodology and identified open challenges. Although faulty transitions could be identified, further research has to investigate the issues regarding mapping, interpreters, and automatic transfer from proprietary logic to behavior descriptions.

The methodology can also be further developed to test and improve resilience for RMS. Resilience is often considered on higher levels of the automation pyramid (ERP- or MES-level), especially for redistributing manufacturing jobs to different machines or supply chains. The methodology could be used for validating and testing stress scenarios which is an essential aspect of resilience in manufacturing as proposed by Weber et al. [21].

References

[1] Zhang L, Qiang Cai Z, Joo Ghee L. Virtual Commissioning and Machine Learning of a Reconfigurable Assembly System. 2020.

[2] Kritzinger W, Karner M, Traar G, Henjes J, Sihn W. Digital Twin in manufacturing: A categorical literature review and classification. IFAC-PapersOnLine [Internet]. 2018;51(11):1016–22. Available from: https://doi.org/10.1016/j.ifacol.2018.08.474

[3] DIN SPEC 91345 - Referenzarchitekturmodell Industrie 4.0 (RAMI4.0). DIN. 2016;(April):1–40.

[4] VDI/VDA Cyber-Physical Systems: Chancen und Nutzen aus Sicht der Automation. VDI/VDE [Internet]. 2013; Available from: https://www.vdi.de/ueber-uns/presse/publikationen/details/cyber-physical-systems-chancen-und-nutzen-aus-sicht-der-automation

[5] DIN EN 61131-1 Speicherprogrammierbare Steuerungen - Teil 1: Allgemeine Informationen. DIN EN 61131-1. 2004;

[6] Thramboulidis K. IEC 61499 vs. 61131: A Comparison Based on Misperceptions. J Softw Eng Appl [Internet]. 2013;06(08):405–15. Available from: http://www.scirp.org/journal/doi.aspx?DOI=10.4236/jsea.2013.68050

[7] DIN EN Funktionsbausteine für industrielle Leitsysteme - Teil 1: Architektur. DIN EN 61499-1. 2014;(September).

[8] Harel D. Statecharts: a visual formalism for complex systems. Sci Comput Program. 1987;8(3):231–74.

[9] Rupp C, Queins S. UML2 glasklar : Praxiswissen für die UML-Modellierung ; eine praktische Übersicht über die Notationselemente der UML zum Heraustrennen ; Extra: mit kostenlosem E-Book]. 4., aktualisierte... München: Hanser; 2012.

[10] State Chart XML (SCXML): State Machine Notation for Control Abstraction [Internet]. [cited 2022 May 12]. Available from: https://www.w3.org/TR/scxml/

[11] OPC Foundation. OPC Unified Architecture Part 16: State Machines. 2021;

[12] Frühwirth T, Pauker F, Fernbach A, Ayatollahi I, Kastner W, Kittl B, et al. Guarded state machines in OPC UA. IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society [Internet]. 2015. p. 4187–92.

[13] Business Process Model and Notation (BPMN), Version 2.0. 2010 [cited 2022 May 16]; Available from: http://www.omg.org/spec/BPMN/20100501

[14] VDMA (Entwurf) - Datenschnittstelle für automatisierte Fertigungssysteme. VDMA 34180. 2010;

[15] Wallner B, Trautner T, Pauker F, Kittl B. Evaluation of process control architectures for agile manufacturing systems. Procedia CIRP [Internet]. 2021 Jan 1 [cited 2021 Dec 17];99:680–5. Available from: https://linkinghub.elsevier.com/retrieve/pii/S221282712100384X

[16] Pauker F, Mangler J, Rinderle-Ma S, Pollak C. centurio.work - Modular Secure Manufacturing Orchestration. 2018.

[17] Biffl S, Lüder A, Meixner K, Rinker F, Eckhart M, Winkler D. Multi-view-Model Risk Assessment in Cyber-Physical Production Systems Engineering. [cited 2022 Mar 15]; Available from: https://orcid.org/0000-0002-6409-8639

[18] Mei K-J, Lee R-S. Collision detection for virtual machine tools and virtual robot arms using the Shared Triangles Extended Octrees method. Int J Comput Integr Manuf [Internet]. 2016 Apr 2 [cited 2022 Apr 25]; 29(4):355–73. Available from: http://www.tandfonline.com/doi/full/10.1080/0951192X.2015.1033755

[19] da Silva AR, Paiva ACR, da Silva VER. A test specification language for information systems based on data entities, use cases and state machines. Commun Comput Inf Sci [Internet]. 2019 [cited 2022 Apr 25]; 991:455–74. Available from: https://link.springer.com/chapter/10.1007/978-3-030-11030-7_20

[20] Leitão HAS, Rosso RSU, Leal AB, Zoitl A. Fault Handling in Discrete Event Systems Applied to IEC 61499. IEEE Int Conf Emerg Technol Fact Autom ETFA. 2020 Sep 1;2020-September:1039–42.

[21] Weber M, Brinkhaus J, Dumss S, Henrich V, Hoffmann F, Ristow GH, et al. EuProGigant Resilience Approach: A Concept for Strengthening Resilience in the Manufacturing Industry. Submit to Procedia CIRP. 2022;00.