Development of a Graph-Based Translation From BPMN to Executable Sequences for Industrial Robotic Systems

Christine Zeh[1], Katrin Clauss[1], Maximilian Papa[1,2] and Sebastian Schlund[1,2]

***Abstract*— The demands regarding high mix, low volume manufacturing and faster product life cycles make flexible production indispensable. Collaborative robots are widely regarded as an enabler for this flexible production. Further, they also achieve the goal of human-centered production envisioned by Industry 5.0. However, its installation requires significant efforts by skilled specialists and robotics experts for robot programming. In order to improve accessibility for shop-floor workers, the focus in previous works lies on the combination of graphical/function-based and declarative programming that supports fast reconfiguration. The business process model notation (BPMN) was used for the user input of hardware-independent robot skills. Further, the so-called SAMY-Core was developed to generate control commands for the respective hardware. Based on these two components, this work focuses on the concluding translation of BPMNs to machine executable processes as the final component within the SAMY framework to finalize its entire pipeline from the user input to the hardware-specific code execution. For the translation, the SAMY-BPMN is processed to a graph, which contains all skills and can return the correct following actions by means of states of the robot system. As a result, it is shown that this translation and thus the entire pipeline is feasible, allowing non-expert users to change the system both quickly and easily.**

I. Introduction

Increasing global competition, shorter product life cycles and individual customer requirements demand a high degree of flexibility in production [1]. Conventional production systems are proving to be too inflexible in this case, and thus, the digitalization of the components and intelligent automation (Industry 4.0) is seen as an enabler. While in the beginning complete automation was sought to achieve this flexibility, it quickly became clear that replacing the human is not considered viable [2]. Thus, human-centered production is established as one of the three pillars of the so-called Industry 5.0 by the European Commission [3].

Collaborative robots (cobots) are widely regarded as one of the enabler for these flexible production requirements [4], [5]. Unfortunately, the reconfiguration of a human-cobot workspace still bears various challenges. Robot experts are usually still needed for programming, and safety would also have to be determined again by safety experts after each adaptation [6]. However, today's shortage of skilled workers represents a significant challenge in this area [7].

The research project "SAMY" aims to address this simplification of accessibility by automating the modification process in control programs of industrial collaborative robotic systems. Preliminary work in this project found that the combination of graphical/function-based (e.g., moving and connecting blocks containing tasks like "pick&place" in a two-dimensional working space) and declarative descriptions (where the work systems' process chain is described with the mentioned blocks and not directly programmed in machine code) form the most effective and robust method for reconfiguring the work system from a user-centric perspective [8]. From these graphical descriptions, the specific robot code must be generated subsequently. As a result of this insight, the SAMY-Editor and the SAMY-Framework (SAMY-Core) [9] were created. In terms of user-centric programming and controlling robotic systems, the SAMY-Editor operates as the frontend using BPMN as the graphical user interface. Whereas the SAMY-Core acts as the backend to the robot for generating the machine code and the final control commands (see figure 1). However, both approaches, using BPMN as well as states and actions for processing, cannot be merged trivially. A middleware between the BPMN and the core is therefore needed. In addition, the translation must somehow process the BPMN, to provide the right work system sequence for generating the corresponding robotic control commands.

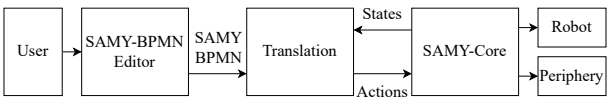

Fig. 1. Simplified framework for declarative programming and its translation to control commands for interaction with periphery

For this reason, this paper describes the process of getting from a simple graphical user notation to the machine-readable representation of it. This middleware is needed to ensure the whole SAMY pipeline (shown in figure 1) from the user input to the actual signal- and movement-conversation on the robotic system. For that, the processing of the graphically described states is needed to ensure a sequence of actions for controlling a robotic system. The development necessary for this, to make BPMNs executable, shows a new approach independent of SAMY.

*This project was funded by the Federal Ministry for Climate Protection, Environment and Energy, Innovation and Technology (BMK), and carried out within the framework of the programme "Production of Future" under the grant agreement number "877362" within the project "SAMY – Semi-Automated Modification in Control Programmes of Industrial Collaborative Robotic Systems".

[1] Fraunhofer Austria Research GmbH, Theresianumgasse 7, 1040 Vienna, Austria

[2] TU Wien, Institute for Management Science, Theresianumgasse 27, 1040 Vienna, Austria

II. State of the Art

Simplifying programming and configuration of industrial robots is a field of ongoing research, in particular by adding an abstraction layer [10]. The main idea of this abstraction is, that many tasks can be represented by a very small number of predefined skills (e.g., "move to <position>" and "pick up <work piece>"). Thus, these skills represent a hardware independent solution of programming a robot, where the available skills are defined by the robot's hardware and its sensors. Based on that, the challenge is combining standardized abstractions and getting from simple user notation to the machine-readable representation of it.

A. Abstraction Layers and Skills-based programming

Hoyos et al. [11] introduce a definition and management of skills, which can be accessed through some user interface. The ability to start one skill at a time, allows for a simple parsing of the abstraction to the machine readable code. The usage of Google Blockly [1] to simplify the programming by adding an abstraction layer is reflected upon in [12] and [13]. In this instance, the parsing of the robot application created through the abstraction layer is bypassed by directly mapping robot specific source code to the Blockly blocks. Thomas et al. [14] uses the Unified Modeling Language suitable for programming (UML/P) to combine the robot code with a graphical modelling language.

B. BPMN as abstraction layer

The use of BPMN as an abstraction layer to the implementation of skill-based industrial programming is a novel approach. Therefore, no existing research can be used as basis for the conversion and processing of the skills plan. In addition, the cited papers of the last section use their development on a specified hardware. Consequently, the translation to machine readable code is more straightforward.

Dijkman and Van Gorp [15] define rules to rewrite BPMN 2.0 to graphs in the tool GrGen. Raedts et al. [16] developed a translation between BPMN and Petri nets to verify and validate models. Further, a conversion between resilient BPMN and directed graphs was developed by Nordemann et al. [17], to use graph-based search algorithms on the BPMN.

A different approach is to directly simulate BPMNs. Pereira and Freitas [18] describe various tools to simulate BPMN. Pufahl et al. [19] introduce further an extensible BPMN discrete event simulator. In addition, the development of a BPMN extension to enable better discrete event simulation is described by Onggo et al. [20].

The conversion of BPMN to machine readable data structure is part of extended research. However, the transcriptions in the cited papers often use sub-types of BPMNs or specialized tools, which are not applicable in the case of robotic programming. Furthermore, the research in the cited papers is focused on economic problems.

III. Methodology

The targeted translation, from graphical descriptions to machine readable plans applies to the area of human-centered reconfiguration and modification, which is hardly tested in the field of robotic programming. Therefore, an iterative software process model with the focus on prototyping and reuse of software, like Boehm's [21] spiral model as methodology to design, build and maintain the concerning interface is applied. According to Alshamrani [22], the spiral model is suitable for developing highly customizable software due to iterative loops, the high amount of risk analysis, and the ability to react to rapid changes. Exemplary iterative loops for the development of the translation software are summarized in table I and table II.

TABLE I
Development of translation software: Iteration 1 (cf. [21])

Objectives	Translation concept for graphical description to machine readable plan
Constraints	BPMN, XML, Python, directed data structure
Alternatives	Petri net/graph, C++/Python
Risks	False description of data structure, states, or transition condition/high implementation effort
Risk resolution	Literature research
Risk resolution results	Development-, translation concept
Plan for next phase	Implementation of translation concept

Table I shows the translation concept for the graphical description to a machine-readable plan and includes the following standards: The BPMN (Business Process Model and Notation)[2] is a graphical notation for processes and represents the user interface for programming the robotic system. The BPMN acts as an input for the transformation to machine-readable plans. The BPMN standard is overlaid with its own SAMY-BPMN [23]. The transformation requires a directed data structure, for which a standard graph structure[3] is used as a connection component between the BPMN as user input and the SAMY-Core [9].

TABLE II
Development of translation software: Iteration 2 (cf. [21])

Objectives	Lossless representation of BPMN to graph (graph builder)
Constraints	BPMN, XML, graph, Python
Alternatives	Standard library BPMN to graph/develop particular translation script + NetworkX
Risks	High implementation effort, loss of information
Risk resolution	Internal discussion, literature research
Risk resolution results	Translation class from BPMN to graph with processing of states & actions
Plan for next phase	BPMN processing (graph planner)

[1] https://developers.google.com/blockly
[2] https://www.iso.org/standard/62652.html
[3] https://www.maths.ed.ac.uk/ v1ranick/papers/wilsongraph.pdf

Table II breaks down the first implementation of the translation concept involves the development of a graph builder, which includes the following standards: Storing the BPMN to XML (Extensible Markup Langage)[4], formatting it to DOM specification (Document Object Model)[5] via the library XML.DOM[6] and then processing the document via Python script to generate the graph is proceeded instead of using a standard library for building a graph, due to the non conventional overlay of the BPMN. For the management of the created graph the library NetworkX[7] is used. The result is a translation class from the BPMN to a graph with the processing of states and actions.

IV. Development

The translation and processing of BPMNs are parts of the connection component between the user interface and the backend core of SAMY.

Figure 2 shows the architecture of the referred connection component defined as controller, and the already implemented SAMY interfaces (SAMY-BPMN Editor, SAMY-Core). The implementation aspect of this work focuses on the controller, specifically on the development and integration of the so-called graph planner, as well as the mapper. A further implementation step is the incorporation of the already implemented interface to the SAMY-Core. As figure 2 shows, the graph planner and the mapping components are direct parts of the controller, which is invoked through the specified interface. The interface and arrows illustrate definitions for the transition of the already developed SAMY parts to the controller. These definitions, namely the use of the SAMY-BPMN on the one hand and the use of states and actions on the other hand, are the constraints for this development. The task can therefore be described as follows: A BPMN is loaded and for each discrete event step, the program is aware of the current action to be executed and whether the present state meets the transition conditions for moving on to the next action. In the development phase only the graph planner is of interest, since the aim of the mapping, described in greater detail in the next section, is the simple parsing of the SAMY specific state and action definitions.

[4]https://www.w3.org/standards/xml/core
[5]https://www.w3.org/TR/WD-DOM/introduction.html
[6]https://docs.python.org/3/library/xml.dom.html
[7]https://pypi.org/project/networkx/

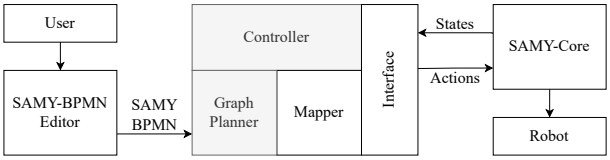

Fig. 2. Architecture of the controller and the surrounding SAMY interfaces

A. Approach for developing the middleware

The approach for the implementation of the two main parts, namely the BPMN transition and the BPMN processing, is dependent on two questions:

- BPMN Translation: How can the BPMN be stored without loss of information?
- BPMN Processing: How can the translation be simulated?

SAMY-BPMN only uses flow objects and connecting objects, which are used to describe a flow of connected activities. Since a flow can contain junctions and loops, a graph-based data structure is necessary to gain a lossless representation. The decision was made for a directed graph, which is processed further. Gateways and variable manipulation tables (VMT) are removed as nodes, to obtain a representation containing only the action-based nodes. The removed information is integrated as internal functionality of the nodes and edges. The nodes fulfil the task of managing the internal variables and returning its action. The edges are opened after checking the received state and the internal variable container. An exemplary depiction of the conversion can be seen in figure 3.

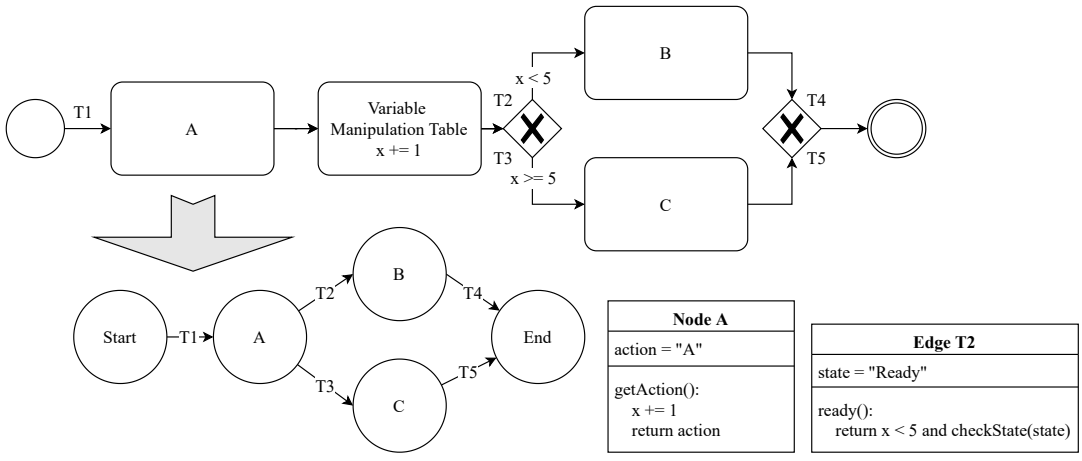

Fig. 3. An exemplary depiction of the conversion between BPMN and directed Graphs with its internal class functionality. The internal functionality of the graph objects is described in pseudo code.

The logic of the process is stored in the objects of the graph, which leads to easy processing of the graph. The initial current node is the start process. Then, at each event, every originating edge of the current node is provided with the current state list. After processing the current state list, the referring edge returns whether the requirements are met for moving on to the next node. Only if the corresponding edge is open, can the next node be reached. If the current node moves forward to the next node, the action request is invoked for this node, which also updates the variable container. As a result, a corresponding skill (e.g., pick&place) is executed on the robotic system.

B. BPMN Translation

A BPMN model is stored as an XML file, which can be processed using a DOM. This is used to store each SAMY-BPMN block type as node and each transition as edge. The result is a function-less graph with the correct connections of the BPMN. The nodes and edges both contain an object which will handle the additional functionality added by gateways and VMTs. Furthermore, it stores the action type of the nodes and the transition state of the edges. As a next step, each gateway and the VMT has to be reworked to move the functionality into the process nodes. After each transformation, the redundant gateway nodes are removed from the graph. This is realized on the basis of the following rules:

- The VMT is stored in a class and the variables are updated when the predecessor is visited within the processing step.
- The conditions for the exclusive gateway are stored in a list of tuples which get checked to open the edges.
- The combination of edges closing the parallel gateway is stored for each involved edge, to ensure each transition is opened, only if all parallel edges are open.
- The loopback gateway does not need extra functionality, therefore solely the predecessor and successor are linked.

C. BPMN Processing

The BPMN contains three blocks which are of special importance for the processing. The start block defines the initialization point of the processing, the end block defines when the processing is finished and the variable container includes all variables needed for the processing with their initial values.

With the start node set as the current node, the process checks on each state event, if the edges are open. If this is the case, the current node moves forward, updates the variable container and returns the action name. Due to parallel gateways, multiple current nodes are possible, which have to be managed collectively. To close a parallel gateway, the list of the parallel edges involved is compared to all linked open edges of current nodes. If they are a subset, the associated current nodes can move forward and merge.

D. Integration in SAMY

The core is limited regarding the user friendliness of describing the actions and states. Therefore, a separate definition for the creation of the BPMNs can be chosen, to facilitate the description of the actions and states. This definition must then be linked to the core actions by a mapping file. The mapping is stored as a YAML file, which assigns a SAMY value to each controller specific value. The mapper can access this file and then map the BPMN actions to the SAMY actions ”on the fly”.

Since there are no transition states within the BPMN, own states must be specified based on the resources of the previous node. For example, a resource could be the robot itself or integrated sensors and actuators. Each resource is defined with its state ”Resource:Ready”, to clarify whether a resource is idle. The state description ”Ready” is arbitrary and could be anything as long as it is used within the mappings. These mappings are content of the YAML file, which stores the correct translation to the SAMY states.

During a triggered state event the mapper is executed twice (see figure 4). Firstly, it is executed at the beginning to parse

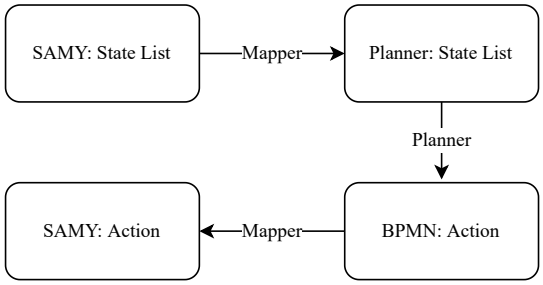

Fig. 4. Mapping pipeline from the SAMY state list to resulting SAMY action.

the SAMY state list to the defined states of the graph planner (planner state list). The graph planner can then work with the mapped states and the actions defined through the BPMN. Secondly, the mapper is executed at the end to converts the resulting BPMN actions to the SAMY actions.

V. Results and Discussion

In order to evaluate the developed translation and mapping from graphical descriptions to machine readable plans, an experimental simulation-based use case was generated for testing the developed controller and the holistic SAMY pipeline. Executing the graphically described process within an exemplary use-case shows the translation from declarative programming and reconfiguration to skill-based processing of work tasks, for generating robotic applications.

The BPMN of the realized use-case is shown in figure 5. The robotic task is to sort five objects into two boxes, with the use of a camera to detect the objects. Overall, ten objects should be moved, after this, the program stops.

The individual blocks in the BPMN are the robotic tasks (e.g., Robot:move pick pose) to be implemented. Those tasks are described as skills within the SAMY context. Loops and branches defined in the BPMN describe the logic of the entire

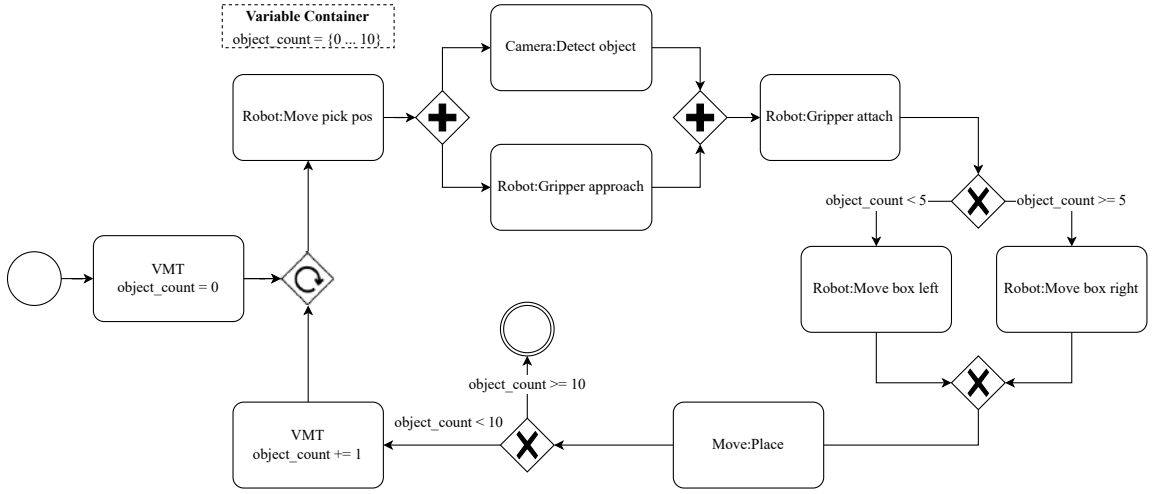

Fig. 5. SAMY-BPMN of exemplary use-case for picking and sorting objects

robotic process. For a logical flow, additional variables are needed, which are defined in the variable container with their initial values. In this use-case such an additional variable is utilized to manage the number of objects already sorted (e.g., object_count < 5). The increasing of the variable is executed by the VMT.

A direct comparison of the BPMN (Figure 5) and the graph (Figure 6) shows that on the one hand the skills still exist as nodes, but on the other hand loop gateways are no longer depicted in the graph. The visualization of exclusive gateways is also omitted and instead represented as logical conditions within the edges. Parallel gateways only appear as conditions for the parallel edges that are converging to the same node. An exemplary progression in the graph from the ”Start” node to the ”Robot:Move pick pos” node is depicted in Figure 7, which corresponds to a controller call triggered by a state change. The first block ”[Robot_RobotUR5_CRCLStatus=1, InformationSource_Camera_Status=0]” describes the physical robotic system, based on a SAMY state list. In this case, unlike the camera, the robot is idle and is waiting for the next instruction. The mapper translates the status of the hardware to a format readable by the planner. The planner checks the received state against the existing graph and returns the action of the next node ”Robot:Move pick pos” if the states match. Lastly the returned action is mapped back as SAMY action ”Move-UR5-Pick” to be processed by the SAMY-Core.

The result of the experimental simulation-based use case is the correct traversing of the graph from start to finish. All states and transition constraints are processed and a SAMY skill is invoked at each node transition, leading to a sequence of skills that controls the hardware and executes the corresponding robotic process. Thus, the feasibility of the simplified reconfiguration of a work system using the SAMY pipeline is demonstrated, where the work system can now be easily modified by drag & drop of the individual BPMN blocks of the process chain (seen in figure 5).

VI. Conclusion and Future Work

Market demands like high mix, low volume manufacturing and faster product life cycles will require a more flexible production system (e.g., human-cobot work system).

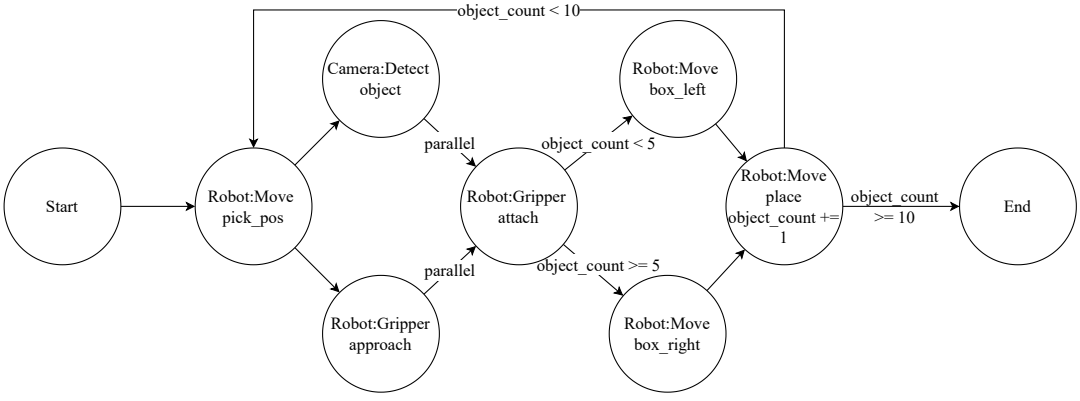

Fig. 6. SAMY-BPMN translation into directed graph of exemplary use-case for picking and sorting objects. For easier readability, the class functionality is written to the corresponding edge or node.

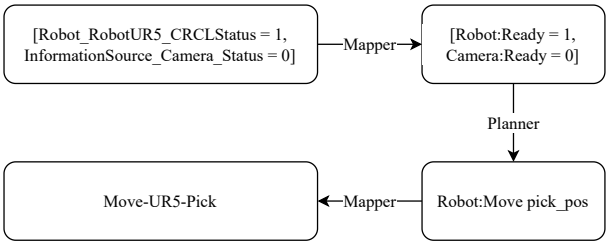

Fig. 7. The skills in the BPMN are mapped to SAMY skills in the mapper.

However, the reconfiguration of such systems is still too complicated and robotics experts are needed. Therefore, SAMY focuses on the simple programming and reconfiguration of these work systems. A (non-) expert creates a simple graphical/function-based SAMY-BPMN as declarative programming and the SAMY-Core creates machine-readable code from it. As a contribution to SAMY, this paper shows the development of the missing connection component between the SAMY-BPMN and the SAMY-Core. The developed graph planner generates a graph containing all skills from the SAMY-BPMN and its sequence. Further, the controller maps the correct skills from the BPMN to the SAMY specific skills, knowing the used hardware. Finally, the interface to the SAMY-Core is done by giving it the actions in the right sequence according to the given states of the robot system. The entire SAMY pipeline was evaluated by an experimental simulation-based use case, which showed that a translation from hardware-independent SAMY-BPMN to specific robot skills was done, meaning that a non-expert could easily reconfigure the work system using the graphical SAMY-BPMN.

Since the use case is solely simulation-based, for further in depth evaluation the experimental use case is implemented in an industrial cobot-application, testing the translation as part of the holistic SAMY pipeline and the benefiting ease of reconfiguration in real-world conditions. In addition, the translation of SAMY-BPMN could be generalized to standard BPMN to use the presented approach in a variety of different applications. Thus, BPMNs that are not cobot-specific could be translated into executable graphs using this method by adapting the SAMY-specific classes. In addition, the presented translator can be made more intelligent (e.g., states as exclusive gateways or swimlanes), so that also more complex logics can be built.

REFERENCES

[1] I. Hanschke, *Digitalisierung und Industrie 4.0 - einfach und effektiv: Systematisch und lean die Digitale Transformation meistern*. Carl Hanser Verlag GmbH & Co. KG, 2019.

[2] A. Kolbeinsson, E. Lagerstedt, and J. Lindblom, "Foundation for a classification of collaboration levels for human-robot cooperation in manufacturing," *Prod. Manuf. Res.*, vol. 7, no. 1, pp. 448–471, 2019.

[3] European Commission, "Industry 5.0." `https://ec.europa.eu/info/research-and-innovation/research-area/industrial-research-and-innovation/industry-50_en` (Accessed: 02.02.2022), 2020.

[4] A. Weiss, A.-K. Wortmeier, and B. Kubicek, "Cobots in industry 4.0: A roadmap for future practice studies on human–robot collaboration," *IEEE Transactions on Human-Machine Systems*, vol. 51, no. 4, pp. 335–345, 2021.

[5] C. Schmidbauer, S. Schlund, T. B. Ionescu, and B. Hader, "Adaptive task sharing in human-robot interaction in assembly," in *2020 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pp. 546–550, 2020.

[6] C. Fischer, M. Steiner, M. Neuhold, M. Papa, A. Markis, and S. Schlund, "An investigation of the measurement of transient contacts in human-robot interaction," in *Advances in Service and Industrial Robotics*, pp. 547–555, Springer International Publishing, 2022.

[7] International Federation of Robotics, "Next generation skills-enabling today's and tomorrow's workforce to benefit from automation," *International Federation of Robotics (Positioning Paper)*, November 2020.

[8] T. Komenda, J. B. Garcia, M. Schelle, F. Leber, and M. Brandstötter, "Sustainable utilization of industrial robotic systems by facilitating programming through a human and process centred declarative approach," *International Conference on Competitive Manufacturing (COMA)*, 2022.

[9] J. B. Gracia, F. Leber, M. Aburaia, and W. Wöber, "A configurable skill oriented architecture based on opc ua (in submission)," *International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[10] S. Bøgh, O. S. Nielsen, M. R. Pedersen, V. Krüger, and O. Madsen, "Does your robot have skills," in *Proceedings of the 43rd International Symposium on Robotics*, 2012.

[11] J. Hoyos, A. B. Junaid, M. Raheel Afzal, A. Tirmizi, and P. Leconte, "Skill-based easy programming interface for industrial applications," in *2022 IEEE/SICE International Symposium on System Integration (SII)*, pp. 210–217, IEEE, 2022.

[12] M. Winterer, C. Salomon, J. Koberle, R. Ramler, and M. Schittengruber, "An expert review on the applicability of blockly for industrial robot programming," in *25th International Conference on Emerging Technologies and Factory Automation*, pp. 1231–1234, 2020.

[13] D. Weintrop, D. C. Shepherd, P. Francis, and D. Franklin, "Blockly goes to work: Block-based programming for industrial robots," in *2017 IEEE Blocks and Beyond Workshop*, pp. 29–36, 2017.

[14] U. Thomas, G. Hirzinger, B. Rumpe, C. Schulze, and A. Wortmann, "A new skill based robot programming language using uml/p statecharts," in *2013 IEEE International Conference on Robotics and Automation*, pp. 461–466, IEEE, 2013.

[15] R. Dijkman and P. Van Gorp, "BPMN 2.0 execution semantics formalized as graph rewrite rules," *Lecture Notes in Business Information Processing*, vol. 67 LNBIP, pp. 16–30, 2010.

[16] I. Raedts, M. Petkovic, Y. S. Usenko, J. M. E. Van der Werf, and J. F. Groote, "Transformation of BPMN Models for Behaviour Analysis," in *Proceedings of the 5th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems*, pp. 126–137, Science and and Technology Publications, 2007.

[17] F. Nordemann, R. Tönjes, E. Pulvermüller, and H. Tapken, "Resilient process modeling and execution using process graphs," vol. 1375 of *Communications in Computer and Information Science*, pp. 3–23, Cham: Springer International Publishing, 2021.

[18] J. Pereira, A. Freitas, M. Teixeira, A. Correia, H. Adeli, A. Rocha, and L. Reis, "Simulation of BPMN process models: Current BPM tools capabilities," *Advances in Intelligent Systems and Computing*, vol. 444, pp. 557–566, 2016.

[19] L. Pufahl, T. Wong, M. Weske, E. Teniente, and M. Weidlich, "Design of an extensible BPMN process simulator," *Lecture Notes in Business Information Processing*, vol. 308, pp. 782–795, 2018.

[20] B. Onggo, N. Proudlove, S. D'Ambrogio, A. Calabrese, S. Bisogno, and N. Levialdi Ghiron, "A BPMN extension to support discrete-event simulation for healthcare applications: An explicit representation of queues, attributes and data-driven decision points," *Journal of the Operational Research Society*, vol. 69, no. 5, pp. 788–802, 2018.

[21] B. W. Boehm, "A spiral model of software development and enhancement," *Computer*, vol. 21, pp. 61–72, May 1988.

[22] A. Alshamrani and A. Bahattab, "A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model," *IJCSI International Journal of Computer Science Issues*, vol. 12, pp. 106–111, 2015.

[23] T. Haspl, M. Rathmair, M. Papa, M. Hofbaur, and A. M. Tonello, "Software toolchain for modeling and transforming robotic workflows into formally verifiable model representations," in *Austrian Robotics Workshop 2022*, 2022. in press.