A Novel Block-Based Programming Framework for Non-programmers to Validate PLC Based Machine Tools for Automotive Manufacturing Facilities

Wolfgang Koehler and Yanguo Jing

Original citation & hyperlink:
Koehler, Wolfgang, and Yanguo Jing. "A Novel Block-Based Programming Framework for Non-programmers to Validate PLC Based Machine Tools for Automotive Manufacturing Facilities." *2018 11th International Conference on Developments in eSystems Engineering (DeSE)*. IEEE, 2018.
https://dx.doi.org/10.1109/dese.2018.00046

ISSN 2161-1351

Publisher: IEEE

A Novel Block-Based Programming Framework for Non-Programmers to Validate PLC Based Machine Tools for Automotive Manufacturing Facilities

Wolfgang Koehler
School of Computing, Electronics and Mathematics
Faculty of Engineering, Environment and Computing
Coventry University
koehlerw@uni.coventry.ac.uk

Dr. Yanguo Jing
School of Computing, Electronics and Mathematics
Faculty of Engineering, Environment and Computing
Coventry University
Yanguo.Jing@coventry.ac.uk

***Abstract*—An automotive manufacturing cell typically consists of multiple stations, controlled by a single industrial programmable controller. Design flaws or assembly mistakes are normally discovered during the highly time-constrained integration phase, which leads to time loss and inefficiency. This paper presents a novel domain-specific 'language' to eliminate PLC experts from the testing process, to minimize input from operators and significantly reduce cost. The proposed 'language' was inspired by widely available educational robotic toys, built on a block based programming environment, which allows for intuitive interaction with novice users. A comparison and evaluation study has been carried out to compare the new framework with the traditional process of building equipment for an automotive manufacturing cell. The study has shown that the proposed 'language' not only eliminates the need for PLC experts, in the testing process, but also reduces the time needed for setup and testing by 90%. In addition, the high level of abstraction has decreased the potential for programming errors by 95%.**

I. MOTIVATION

Manufacturing cells, within an automotive body shop, typically consist of several stations [1], which are linked together by a transfer mechanism. These stations are made up of part sensing switches, a few electric or pneumatic actuators and associated position sensors.

Such systems are often built in a modular fashion, where the stations are built, piped and wired as standalone components, independent of their final integration into the cell. Since there is only one PLC (programmable logic controller) controlling the whole cell, there is no simple way of functionally validating each of the stations during the build phase. Testing the stand-alone stations during the build phase however yields many benefits:

- Typically, there is more time available at this point of the manufacturing process than during integration
- Piping and hosing of the tooling can be verified to be correct and without leaks
- Wiring can be verified to be correct
- Settings of I/O field devices can be verified
- Mechanical components can be optimized for best performance
- Sequence of the tooling can be checked for possible interference

Because the testing of the machine tools prior to the integration phase offers clear benefits, some machine builders have set up a 'test PLC' for a preliminary validation. This 'test PLC' requires custom programming by a controls expert. Often however, the testing at the build phase is skipped due to time and personnel constraints.

The proposed block-based 'programming' framework is a novel approach to overcome these constraints. There is no longer the need for a controls expert to validate the tooling. The new framework also provides a user-friendly programming tool that enables non-programmers to perform the testing. The time needed to perform the testing has been decreased as well, which makes this proposal practical for use in day to day operations.

II. RELATED WORK

The approach taken for this paper can be best compared to domain-specific languages (DSL), DSL workbenches and language design for end-user programming, which has been extensively discussed in the works of [2], [3], [4].

However, limited work has been done in the area of enabling users with a limited programming knowledge to operate in an intuitive environment for the quality checking of design flaws and assembly mistakes, prior to the integration phase in automotive manufacturing facilities. This work has filled that gap by introducing a semi-automated graphical domain-specific 'language' that makes the whole process seem more like configuration, rather than programming, to the end user.

The main research question concerning this paper was: 'What could a framework look like that would allow the novice user to perform a functional test on industrial machine tools without any PLC programming knowledge?' It first came to mind, that the user interface must be graphical and intuitive rather than text based. This intuition is also shared by others [5], [6], [7], [8], [9], [10] who have researched methods to make industrial programming more efficient. Daehnhardt and Jing used a semantic web approach to design a software

framework for novice users [11]. Jing et al. proposed an intelligent framework for an intelligent system [12].

Vyatkin [9] states that according to IEC 61131-3, the five PLC programming languages are: 'structured text (ST), instruction list (IL), function block diagrams (FBD), ladder logic diagrams (LLD) and sequential function charts (SFC)'. According to Vyatkin, only FBD and LLD are considered graphical languages according to Vyatkin. For him, SFC is categorised as a language for 'overall configuration'.

The approach of FBD is object oriented. A function block (or as Rockwell calls it 'add-on instruction') can be freely defined as a 'black box', consisting of some internal logic 'hidden' from the user's view and parameters which have to be applied to achieve the desired function. The level of abstraction depends on the creator of the function block. However, it is still to be used within the PLC programming environment in conjunction with some other logic instruction, which to the novice user, can still be intimidating.

The second alternative is SFC. This approach is similar to a flow chart based on actions and transitions. Whenever a transition condition is met, the following action will be executed. As Theorin [13] states, it 'consists of three parts which may be considered separate sub-languages: the FC language (graphical), the action language (textual), and the condition language (textual).' This leads to the conclusion that SFC will also not be intuitive to the novice user. The improvements proposed by Theorin concern additional functionality rather than user friendliness.

Some papers on the other hand [14], [15], [16], [17] propose a graphical approach based on the UML notation. It consists of a structure editor, a behaviour editor and I/O mapping utility. To validate the concept Obermeier [15] chose: 'a sample of 168 participants consisting of five school classes (three beginner second year classes, two intermediate third year classes) from a vocational school for production engineering in Munich with specialization in mechatronics, as these apprentices form the relevant group of novice and intermediate PLC programmers in Germany.' The results clearly show that all the participants performed better using the proposed graphical approach versus the standard PLC programming languages.

The use of domain specific programming blocks that just need to be parameterized, is also proposed in the works of [6], [18], [8]. Besides the definition of common programming blocks several papers [19], [14] also point out the need for a common data format that allows for a seamless, automated integration of the different design processes. Because of its platform independence, the choice is often some variation of XML.

In recent years, many robotic educational toys have emerged, which rely on a block based interface for their programming environment. These toys are often rated for children of 6 years and older, which would lead to the assumption that the programming environment must be abstract and intuitive enough for a novice user. The research performed by [20], [8], [21] confirms this suspicion. Price et al. [20] state: 'We found that while the interface did not seem to affect users' attitudes or perceived difficulty, students using the block interface spent less time off task and completed more of the activity's goals in less time.'

III. THE FRAMEWORK

A. Considerations

The proposed framework was to be tested for tooling following the global Opel / Vauxhall controls design standards. Therefore, the following hardware components were chosen:

- Rockwell Controllogix PLC
- Rockwell Devicenet module
- Siemens IPC based touch screen HMI
- Power supply to power PLC and Devicenet Network

Furthermore, the following limitations where outlined based on the above design standard:

- maximum of 10 modular I/O nodes numbered 30 39
- maximum of 10 valve nodes numbered 40 49
- maximum of 240 sequence steps within a station

B. Methods

The schematic structure of the test equipment shown in Figure 1 has been proposed. A key novelty of this structure is to eliminate the requirement for end users to write any PLC code to control the machine tool. The overall process can also be seen in a video shown in Figure 2 (available on https://youtu.be/AguOsaejFY0).

Step 1: a XML description was defined for each of the above blocks. The definition for a manifold with one cylinder is shown in Figure 3.

Since the testing equipment should provide an intuitive way for configuration, it was decided to use Google Blockly as an interface. According to Google (2016), 'Blockly [is] a library that adds a visual code editor to web and Android apps. The Blockly editor uses interlocking, graphical blocks to represent code concepts like variables, logical expressions, loops and more. It allows users to apply programming principles without having to worry about syntax or the intimidation of a blinking cursor on the command line.'

The blocks also allow for the input of additional parameters. Each of the blocks is associated with a textual representation of its function. In its original intent, this would be a code snippet in the chosen programming language. As output, Blockly generates a file which links all the code snippets together based on how the blocks are linked with each other. XML was selected to transfer the data from the Blockly configurator to the HMI screen.

Step 2: Considering the physical layout of a manufacturing station. It was determined that it could be broken down to the following components: station, part sensors, manifolds, valves, cylinders, position sensors and input blocks.

In order to describe the sequence of the tooling as well, the following additional components are needed: action description and complete conditions to prevent tooling interference or to describe additionally needed automatic conditions. This leads to the creation of the following Blockly blocks (Figure 4):

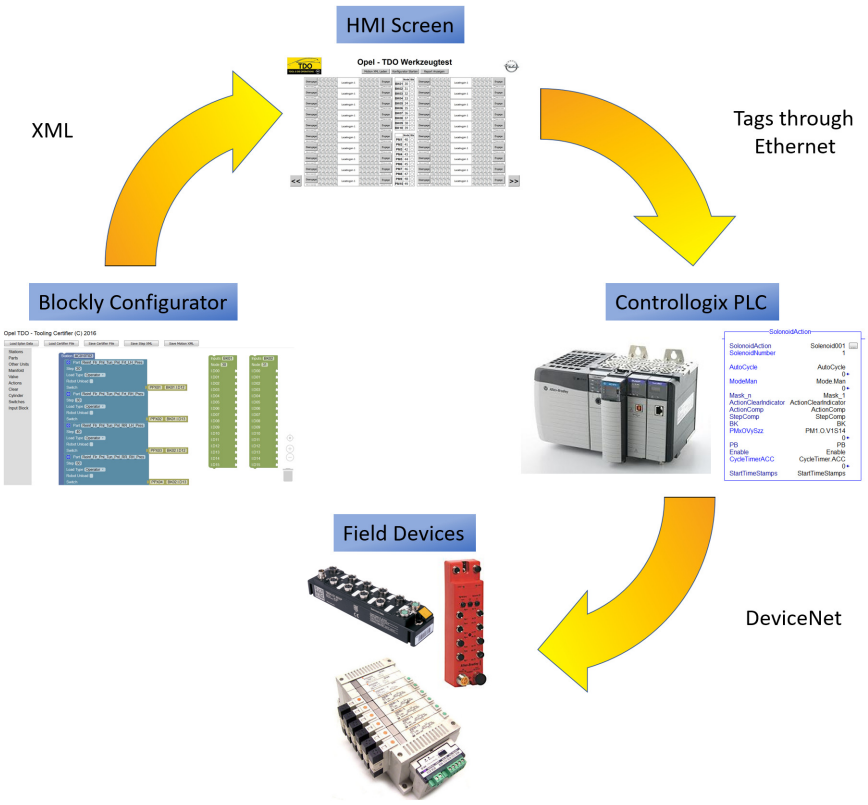

Fig. 1. The graphical programming framework for PLC based machine tool validation

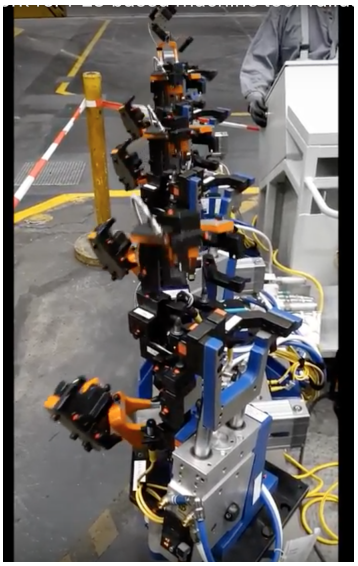

Fig. 2. Video Demo (available on https://youtu.be/AguOsaejFY0)

The initial intent of the system was that the operator would pick the appropriate Blocks from the Blockly toolbar and arrange them according to the hardware drawings. This could be rather cumbersome for a bigger station.

However, there are several files in .csv format available from the hardware design package, which are used to engrave the tags to be mounted to the tool. These tags essentially include most of the information needed to describe the station. Therefore, an additional function was created that would parse through these .csv files and arrange the corresponding blocks on the work space. All that is left for the operator at this point, is to enter the step numbers and to assign the inputs for the part sensing switches.

Step 3: XML import into the HMI screen. The HMI uses a script to import the Blockly XML output file. During this process, the XML file is parsed and the HMI is set up accordingly. Buttons and indicators are correctly labelled and assigned the appropriate motion tags. In addition, the masks will be generated that are needed to communicate the sequential information to the PLC.

Step 4: Interface to the PLC. The PLC contains the motion logic for 120 valves / 240 solenoids. A concept based on masks was developed, since we do not know ahead of time which switches are to be assigned to a solenoid and which conditions must be met for the clear and auto rungs. These masks are generated in the HMI and are transferred to the PLC via 'Power Tags'. The masks for every motion, consisting of 35 integers, is set up as follows (table I):

Step 5: PLC Integration. On the PLC side, an add-on instruction has been created which is used once for every motion. It's SolenoidAction, SolenoidNumber, Mask_n and

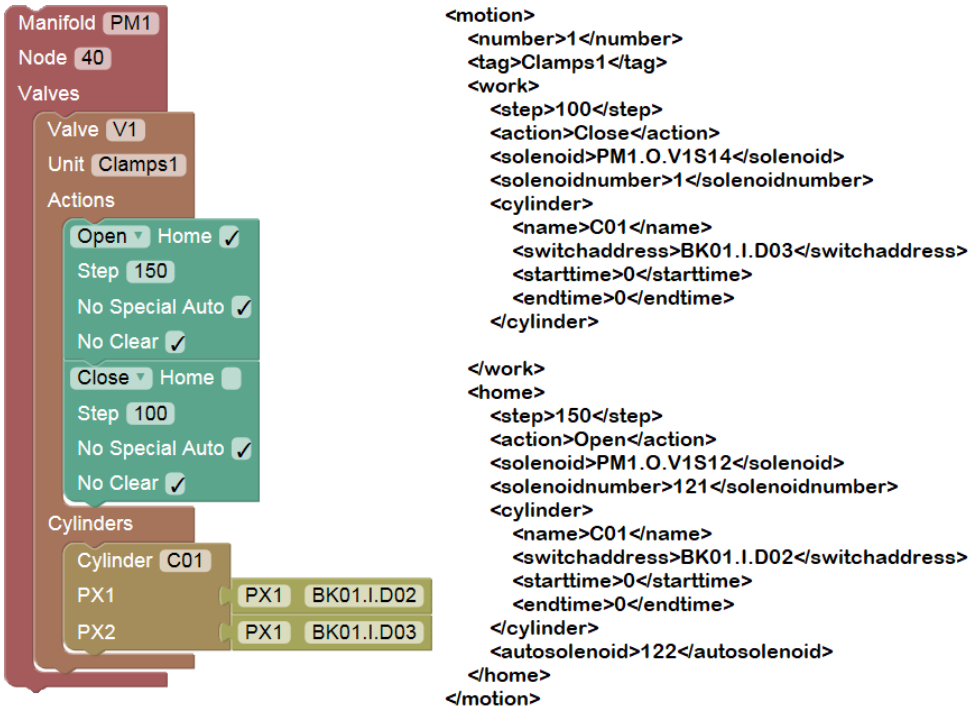

Fig. 3. : Blocks and their XML description

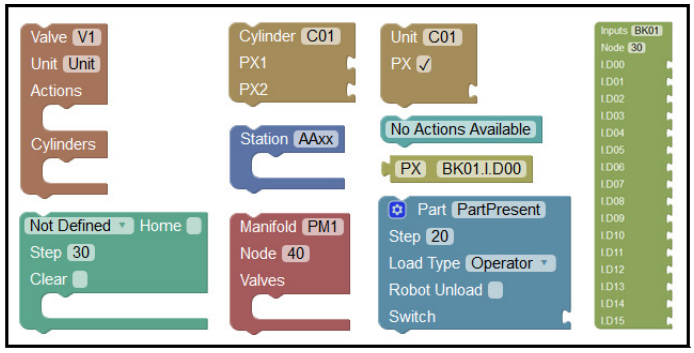

Fig. 4. : Custom Blocks created

PMxOVySzz parameters are unique.

The complete Signal is determined within this add-on instruction. If the block status (BKSts) is considered as 'a' and the mask for that block (BK) as 'b' then the motion is considered complete when the logic in (1) is true.

$$(a[1] \vee b[1]) \wedge (a[2] \vee b[2]) \tag{1}$$

Step 6: In the next step the clear signals are checked with this logic to make sure there are no interference. If ActionComp is considered as 'c' and the clear masks as 'd' then there is no interference present when the logic in (2) is true.

$$(c[0] \vee d[0]) \wedge (c[1] \vee d[1]) \wedge \ldots \wedge (c[16] \vee d[16] \tag{2}$$

TABLE I
MASKING BITS DEFINED

BKMask (Int)	Determines which two BKs are used for the inputs of this motion. Only bits 0 9 are used. Every bit represents one BK.
BK1 (Int)	Determines which inputs of the first BK are used for this motion. Only bits 0-7 are used. Every bit represents one input point.
BK2 (Int)	Determines which inputs of the second BK are used for this motion. Only bits 0-7 are used. Every bit represents one input point.
ClearBits (15xInt)	Determines which solenoid number is a clear condition. Every bit represents one motion that needs to be completed in order not to interfere with this motion.
AutoBits (15xInt)	Determines which solenoid number is an auto condition. Every bit represents one motion that needs to be completed in order for this motion to occur.
PB (Int)	Holds the PB number for this motion as a decimal value.
Enable Solenoid	Holds the solenoid number of the enable solenoid as a decimal value.

The same approach is taken for the auto bits.

Considering StepComp to be 'e' and the auto mask to be 'f', then the auto conditions are met when the logic in (3) is true.

$$(e[0] \vee f[0]) \wedge (e[1] \vee f[1]) \wedge \ldots \wedge (e[16] \vee f[16]) \tag{3}$$

The logic is completed by two rungs that turn on the enable valve as well as the action solenoid where the byte and bit are calculated based on the solenoid number. The controls logic

also contains some rungs to determine when the station is in home position in order to reset the complete latches and a rung to determine the mode.

To allow for communication to any combination of possible Devicenet nodes the system only needs to be set up for the maximum allowed Devicenet nodes. If not all the nodes are connected, a fault will be displayed on the Devicenet scanner, which will not however impact the work ability of the system.

IV. EVALUATION

The test stand consists of a console, to which a touch sensitive 23" display with integrated industrial PC, is mounted. The console also houses an Allen Bradly controls logic PLC including Devicenet scanners, which provide the interface for the tooling to be tested. Last but not least, for safety purposes, an e-stop push button and a two hand start are incorporated as well. For communication to the tooling the Devicenet scanners were set up to the maximum allowed number of nodes.

This test setup was the basis for performing a tool validation in the traditional way (referred to as the traditional method), using custom software and HMI screens as well as for a second test, based on the newly created domain-specific 'language', in order to highlight it's benefits. The evaluation was done validating a newly built tool with twelve cylinders (24 motions) and 16 part present switches.

After completing the validation, based on the custom software, the frameworks generic PLC logic was loaded into the controls logic PLC, and the Blockly based user interface software was installed on the industrial PC. This setup is a one-time configuration which will not have to be done or altered by the end user.

The whole testing process was completed in less than 16 minutes by a tradesman unfamiliar with controls, using the new domain-specific 'language'. To get the same results using a standalone PLC, a controls engineer took more than 2.5 hours. A detailed comparison is shown in Figure 5. Based on these findings, a 90% reduction in testing time by can be concluded.

For completion, it needs to be mentioned that the comparison was only done on a single (yet typical) tool as it would be a non-value added exercise for the company to test every tool a second time using the more time-consuming traditional approach. Testing has ever since been done with the proposed system however and is consistently completed in less than 30 minutes per tool.

In addition, it has to be considered that a typical programming environment based on IEC 61131-3 provides about 200 different instructions. Every one of those instructions can also be considered a potential opportunity for failure. Since the 'instruction set' within this domain-specific 'language' was essentially reduced to 10, it can be argued that at the same time the potential for programming errors was reduced by 95% as well.

V. CONCLUSION AND FUTURE WORKS

It was found that configuring the station using Blockly was rather intuitive for the machine builder and the model

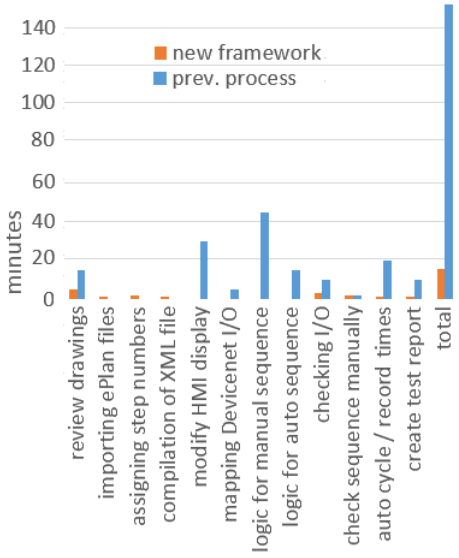

Fig. 5. : Overall Test Results

created could easily be related to the real-world machine setup. Because of this, the time required for preparing the test bed and performing the test could be dramatically decreased. Additionally, the HMI interface guided the user step by step through the testing processes without the need for any specialised training. The automatically generated test report was proved to be valuable tool during the hand-off of the equipment from the modular build department to the integration department. As an additional benefit, it was observed that due to its high abstraction level, the new domain-specific 'language' minimises the chance for programming errors as well.

It can be summarised that the novel framework proposed in this paper will not only lead to a reduction in the man hours required for testing, but will also improve the process of custom machine tool building, while increasing the quality of the product at the same time. Since many of the current PLCs have adopted a tag based programming environment the system can easily be adapted to PLCs from other manufacturers by setting up the tags within the PLC and creating an equivalent code.

The biggest limitation to the proposed framework is the fact that the programming software available from most PLC manufacturers is proprietary. If this were not the case, the work around using masks to control the PLC program would not be needed. Instead, it would be feasible that the configurator could generate a PLC program that could be directly downloaded to the processor. That would potentially open the door to more efficient programming, if customised blocks, suited to the end customer's needs, could be developed. Programming could then really be reduced just to replicating the physical machine layout within Blockly.

ACKNOWLEDGMENT

The authors would like to thank Opel Automobile GmbH and Vauxhall for sponsoring this project and providing the equipment and manpower necessary for the evaluation trials.

REFERENCES

[1] D. F. Sadok, L. L. Gomes, M. Eisenhauer, and J. Kelner, "A middleware for industry," *Computers in Industry*, vol. 71, pp. 58 – 76, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166361515000615

[2] M. Fowler, *Domain-specific languages*. Pearson Education, 2010.

[3] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," *ACM computing surveys (CSUR)*, vol. 37, no. 4, pp. 316–344, 2005.

[4] C. Preschern, A. Leitner, and C. Kreiner, "Domain specific language architecture for automation systems: an industrial case study," in *Workshop on Graphical Modeling Language Development*, vol. 136, 2012.

[5] S. A. Bohner and S. Mohan, "Model-based engineering of software: Three productivity perspectives," in *2009 33rd Annual IEEE Software Engineering Workshop*, Oct 2009, pp. 35–44.

[6] J. Fischer, B. Vogel-Heuser, and D. Friedrich, "Configuration of plc software for automated warehouses based on reusable components- an industrial case study," in *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, Sept 2015, pp. 1–7.

[7] C. R. Magar, N. Jazdi, and P. Ghner, "Requirements on engineering tools for increasing reuse in industrial automation," in *ETFA2011*, Sept 2011, pp. 1–7.

[8] M. Spindler, T. Aicher, D. Schtz, B. Vogel-Heuser, and W. A. Gnthner, "Modularized control algorithm for automated material handling systems," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, Nov 2016, pp. 2644–2650.

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

[10] Y. H. Yin, J. Y. Xie, L. D. Xu, and H. Chen, "Imaginal thinking-based human-machine design methodology for the configuration of reconfigurable machine tools," *IEEE Transactions on Industrial Informatics*, vol. 8, no. 3, pp. 659–668, Aug 2012.

[11] E. Daehnhardt and Y. Jing, "An approach to software selection using semantic web," *IAENG International Journal of Computer Science*, vol. 40, no. 4, pp. 238–249, 2013.

[12] Y. Jing, N. Taylor, and K. Brown, "An intelligent inference approach to user interaction modeling in a generic agent based interface system," in *Proceedings of the 15th European Conference on Artificial Intelligence*. IOS Press, 2002, pp. 103–107.

[13] A. Theorin, *A sequential control language for industrial automation*. Department of Automatic Control, Lund Institute of Technology, Lund University, 2014.

[14] D. Hästbacka, T. Vepsäläinen, and S. Kuikka, "Model-driven development of industrial process control applications," *Journal of Systems and Software*, vol. 84, no. 7, pp. 1100–1113, 2011.

[15] M. Obermeier, S. Braun, and B. Vogel-Heuser, "A model-driven approach on object-oriented plc programming for manufacturing systems with regard to usability," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 3, pp. 790–800, June 2015.

[16] D. Tikhonov, D. Schtz, S. Ulewicz, and B. Vogel-Heuser, "Towards industrial application of model-driven platform-independent plc programming using uml," in *IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society*, Oct 2014, pp. 2638–2644.

[17] K. Thramboulidis and F. Christoulakis, "Uml4iota uml-based approach to exploit iot in cyber-physical manufacturing systems," *Computers in Industry*, vol. 82, pp. 259 – 272, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016636151630094X

[18] C. Mahler, "Automatisierungsmodule für ein funktionsorientiertes automatisierungsengineering," 2014.

[19] E. Estevez and M. Marcos, "Model-based validation of industrial control systems," *IEEE Transactions on Industrial Informatics*, vol. 8, no. 2, pp. 302–310, May 2012.

[20] T. W. Price and T. Barnes, "Comparing textual and block interfaces in a novice programming environment," in *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, ser. ICER '15. New York, NY, USA: ACM, 2015, pp. 91–99. [Online]. Available: http://doi.acm.org/10.1145/2787622.2787712

[21] S. Ulewicz and B. Vogel-Heuser, "Guided semi-automatic system testing in factory automation," in *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*, July 2016, pp. 142–147.