Assembly Workers as PLC Programmers: A Low-Code Development Platform for Engine Manufacturing

Bernhard Schenkenfelder, Raphael Zefferer, Christian Salomon and Mario Winterer

Software Competence Center Hagenberg GmbH, Softwarepark 32a, 4232 Hagenberg, Austria

Abstract

A Low-Code Development Platform (LCDP) empowers domain experts without software engineering training to address their software requirements themselves. Techniques such as visual programming and programming by natural language allow them to create software in a way that is more natural to how they think about their problems, thereby leveraging the low-code paradigm rather than traditional textual code. In this paper, we present an LCDP based on the visual programming paradigm and the considerations that guided its design. The LCDP allows assembly workers to generate Programmable Logic Controller (PLC) code for the assembly devices they operate. Feedback from this industry-academia collaboration suggests that the platform can help to reduce the need for professional PLC programmers and allow them to focus on other important issues.

Keywords

Low-Code Development Platform (LCDP), Programmable Logic Controller (PLC), Structured Control Language (SCL), Assembly Workers, Assembly Devices

1. Introduction

By raising the level of abstraction beyond code, a Low-Code Development Platform (LCDP) empowers domain experts without software engineering training to address their software requirements themselves, bridging the gap between the high demand for software and the low supply of professional developers [1] and increasing productivity and reducing complexity [2]. Techniques such as visual programming and programming by natural language enable users to create software in a way that is more natural to how they think about their problems, as opposed to traditional textual code [3].

2. Industry Context

This paper reports on an industry-academia collaboration centered on *PLC-Flow*, an LCDP designed and intended for assembly workers. The industry partner is a manufacturer of engines for the recreational vehicle industry with a turnover of 1 billion euros and 1,700 employees. Some assembly line stations require custom-built devices that are controlled by Programmable Logic Controllers (PLCs). However, the shortage of PLC software developers poses a challenge. One possible solution, which served as the starting point for the industry-academia collaboration, was to empower assembly workers to program these devices directly using a low-code approach. This issue was first identified and described in [4]. *PLC-Flow* allows assembly workers to generate PLC code for approximately 50 devices per year. The industry partner provided the following use case descriptions to outline the scope of representative assembly devices and their corresponding PLC code.

Joint Proceedings of IS-EUD 2025: 10th International Symposium on End-User Development, 16–18 June 2025, Munich, Germany
Bernhard.Schenkenfelder@scch.at (B. Schenkenfelder); Raphael.Zefferer@scch.at (R. Zefferer); Christian.Salomon@scch.at (C. Salomon); Mario.Winterer@scch.at (M. Winterer)
https://www.scch.at/team/bernhard.schenkenfelder (B. Schenkenfelder); https://www.scch.at/team/raphael.zefferer (R. Zefferer); https://www.scch.at/team/christian.salomon (C. Salomon); https://www.scch.at/team/mario.winterer (M. Winterer)
0000-0001-5129-6268 (B. Schenkenfelder); 0009-0000-5648-9704 (R. Zefferer); 0000-0002-5665-2919 (C. Salomon); 0000-0002-3894-4635 (M. Winterer)

2.1. Use Case 1

Use case 1 consists of four steps and is part of an assembly line, so the assembly device is controlled by another system. The start of the process is determined by the higher-level control of this assembly line and the completion must be reported back. The first step is to place the part into the assembly device, it is finished when two sensors send a positive signal to the PLC. Next, more parts are attached until the rising edge of a sensor signal is detected. Then, to bolt on the attached parts, outputs are set positive to activate a power screwdriver until an input gives a positive signal. When the part is removed from the assembly device, two sensors give a negative signal. The result of the cycle must be signaled to the higher-level control and also indicated by lights.

2.2. Use Case 2

Use case 2 has 10 steps but no higher-level control. First, the part must be placed in the assembly device, which is completed when a sensor gives a positive signal. The next step is to fix the part. To do this, an output must be set to a positive state. The step is completed when two inputs are positive and one is negative. The next four steps are to check the part. These steps are completed when the sensors provide the required signals and are indicated by a green light. Screws are then inserted and tightened. To do this, three outputs are set to a positive state to activate a power screwdriver. When an input signals that this step is complete, the cycle continues with the removal of the part. This step is completed when an input signals a negative state. The next step is to place the part on a tray. A positive signal indicates the end of this step. After that, more screws are inserted and tightened as before. Then, an output is set to a positive state to signal the end of a production cycle.

3. PLC-Flow

This chapter presents the history of *PLC-Flow*, from early design considerations to an initial digital prototype to the platform in its current state.

3.1. Design

The overall idea was to design the platform to reflect the mental model of its users. A mental model is a person's internal representation of reality [5]. This ensures that the platform can be used intuitively, is easy to learn, and optimally supports the way users work. The design process began with an extensive literature review and research phase by the academic partner. The goal was to provide an overview of low-code approaches in order to discuss their respective applicability for the industrial use cases and to select a basis for the implementation of the prototype. Therefore, it was necessary to categorize the approaches.

3.2. Prototype

The first versions of the digital prototype were built using Next.js[1] and React[2] to discuss the main building blocks of the platform and how to interact with them. Figure 1 (left) shows a list of components (sensors and actuators), including their addresses, process steps defined as components and their respective state.

[1] https://nextjs.org/
[2] https://react.dev/

3.3. Architecture

PLC-FLow was built using Angular v17[3] with Angular Material[4]. To generate Structured Control Language (SCL) code, a visitor was implemented based on ANTLR4[5]. A grammar was defined for ANTLR4 to generate the lexer, parser, and the visitor base class. This visitor is included in *PLC-Flow* as a service.

The User Interface (UI) has a column-based layout, guiding the user to work from left to right to complete a process. The first main column lists all available components. As shown in Figure 1 (right), columns can be expanded and collapsed. When the column is expanded, the user can add, edit, or remove components. The second column enables the user to define the program, which is divided into four parts: (i) the initial position of the assembly device, (ii) the reset action, (iii) the steps of the assembly process, and (iv) the actions to be executed when a process completes successfully or unsuccessfully. On the left side of *PLC-Flow*, there are two additional columns to import a CAD drawing of the assembly device, to load other projects, and to define whether the assembly device is part of an an assembly line. On the right, the provided export functionality enables the user to save the program as an archive. The archive contains the generated SCL code and a flow file, which represents the project in the defined grammar. This column also contains a preview of the generated SCL code.

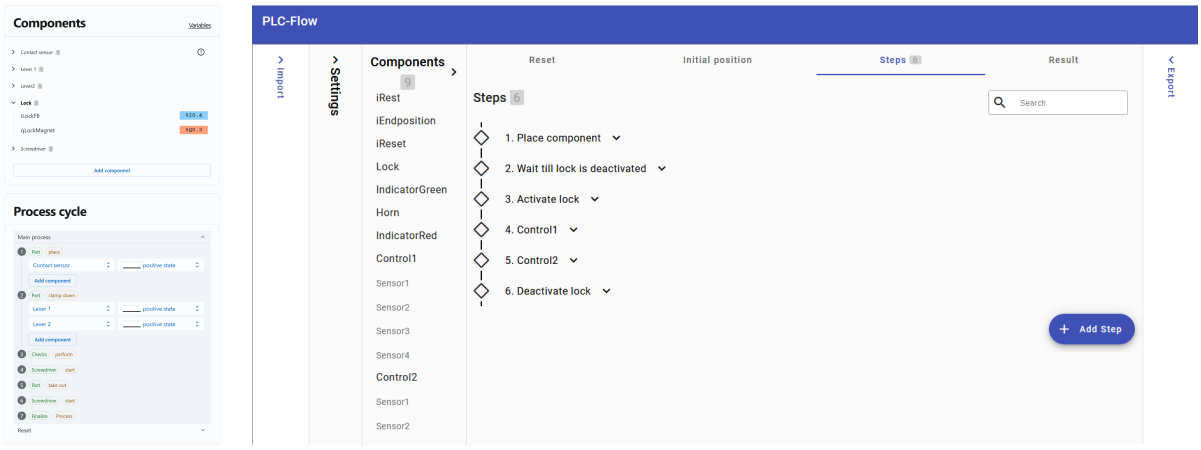

Figure 1: The prototype of *PLC-Flow* shows components, a CAD drawing, and the process steps of a program (left). Overview of *PLC-Flow* and its column-based layout (right).

4. Demonstration

This section demonstrates how a user can create a program using *PLC-Flow*[6], a process that is divided into eight steps.

4.1. Import

First, the user can import the flow file of an existing project, which extracts all the components and process steps. The user can also import a CAD drawing of the assembly device, as shown in Figure 2 (left).

4.2. Settings

In the second step, the user can define if the assembly device is used in an assembly line or before. If it is used in an assembly line, the user can define the inputs and outputs of this line. For the assembly

[3] https://angular.dev/
[4] https://material.angular.io/
[5] https://www.antlr.org/
[6] A public version of *PLC-Flow* is available online at https://plcflow.scch.at.

line case, an input signaling when to start a process and two outputs signaling when the process is completed successfully or not must be configured, as illustrated in Figure 2 (right).

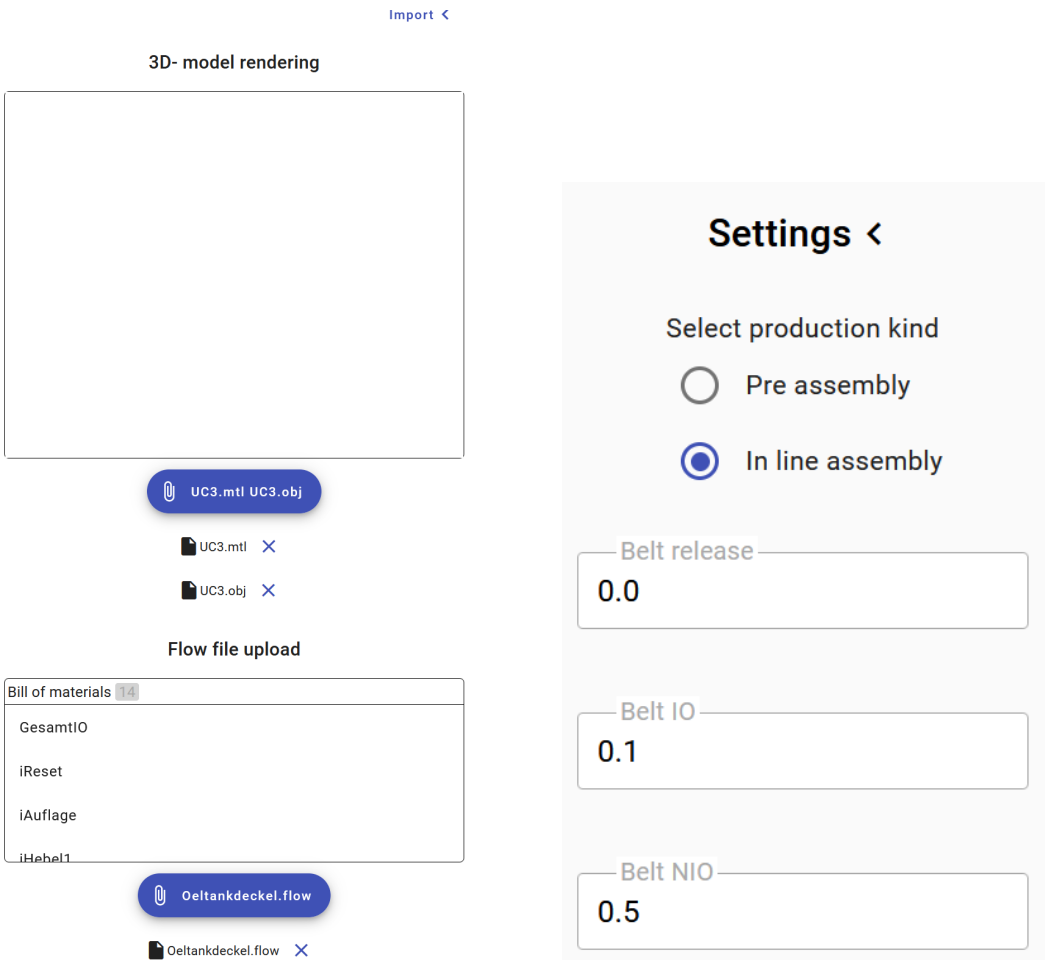

Figure 2: Import of CAD drawings, loading of projects (left) and definition of Settings (right).

4.3. Components

Next, the components need to be assigned a name and multiple inputs and outputs, each with its own name and logical address, as shown in Figure 3 (left). The logical address refers to the physical address used on the PLC.

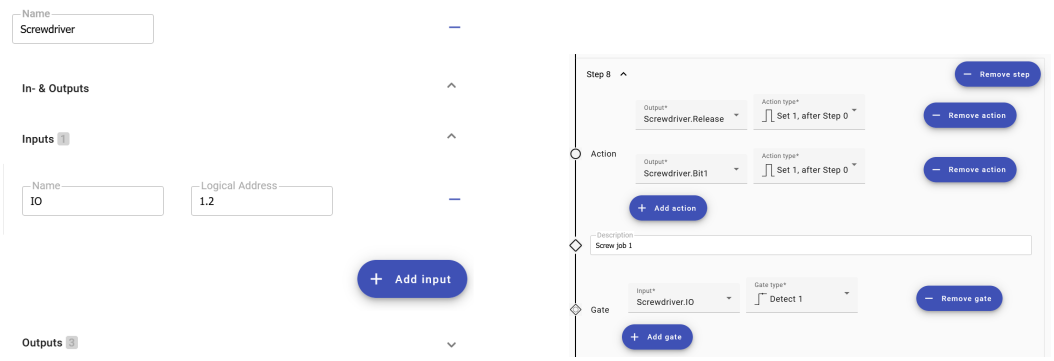

Figure 3: Component definition (left) and definition of a step (right).

4.4. Initial Position

The next step is to define the initial position of the assembly device by selecting the initial states of the required inputs from drop-down menus. In the first drop-down menu, the component can be selected by its name. The second drop-down menu defines the state that corresponds to the initial position. Examples of these drop-down menus can be seen in Figure 3 (right). Each of these conditions is connected by a logical conjunction in the generated SCL code.

4.5. Reset

The user can then define how to reset the assembly device by adding a reset gate and an action. A gate defines the conditions that must be met for the device to be reset. An action defines the state of a PLC output upon reset. These definitions are again made using drop-down menus (see Figure 3, right).

4.6. Process Steps

Then, the steps to be executed for each production cycle can be defined. Each step is indexed, starting with 1, and consists of an action that defines what happens in that step, a description of the step, and a gate. The gate defines the conditions that must be fulfilled for the step to be successfully completed and for the next step to begin. Steps can be rearranged using drag and drop, and expanded and collapsed. When collapsed, only the index and the description of a step are displayed, as shown in Figure 1 (right). Otherwise, the action, description and gate can be edited (see Figure 3, right).

4.7. Result

The final step in defining a complete process in *PLC-Flow* is to define what happens at the end of each cycle by setting the actions to be executed after a successful or unsuccessful production cycle.

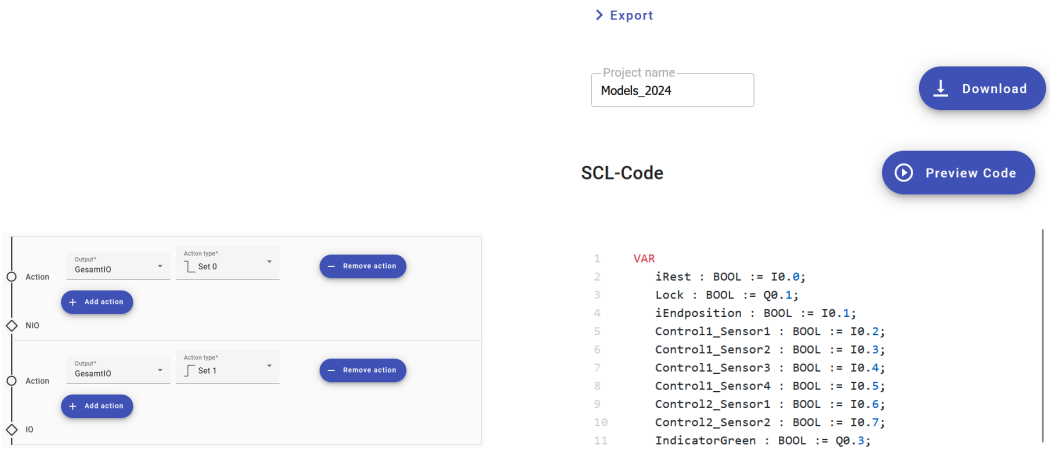

Figure 4: Definition of the result (left) and program export and SCL code preview (right).

4.8. Export

The user can preview the generated SCL code in the export area of *PLC-Flow*. In addition, the user can name and download an archive containing the SCL code and a flow file, as shown in Figure 4 (right).

5. Conclusions

This paper reports on an industry-academia collaboration centered on *PLC-Flow*, an LCDP designed and intended for assembly workers to generate Programmable Logic Controller (PLC) code.

Throughout the design process, reviews with assembly workers–who will be the future users–have confirmed that the implementation aligns with the original goals. This includes the level of abstraction (see Section 3.1) and *PLC-Flow*'s overall usability.

Acknowledgments

The research reported in this paper has been funded by the Federal Ministry for Innovation, Mobility and Infrastructure (BMIMI), the Federal Ministry for Economy, Energy and Tourism (BMWET), and the State of Upper Austria in the frame of the SCCH competence center INTEGRATE (FFG grant no. 892418) in the COMET - Competence Centers for Excellent Technologies Programme managed by Austrian Research Promotion Agency FFG.

Declaration on Generative AI

During the preparation of this work, the author used DeepL Write and DeepL Translate in order to: Text Translation, Improve writing style, and Grammar and spelling check. After using these tools/services, the author reviewed and edited the content as needed and takes full responsibility for the publication's content.

References

[1] R. D. Caballar, Programming without code: The rise of no-code software development - ieee spectrum, 2020. URL: https://spectrum.ieee.org/programming-without-code-no-code-software-development.

[2] J. Cabot, Positioning of the low-code movement within the field of model-driven engineering, in: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, ACM, Virtual Event Canada, 2020, p. 1–3. URL: https://dl.acm.org/doi/10.1145/3417990.3420210. doi:10.1145/3417990.3420210.

[3] M. Hirzel, Low-code programming models, Communications of the ACM 66 (2023) 76–85. doi:10.1145/3587691.

[4] B. Schenkenfelder, C. Salomon, G. Buchgeher, R. Schossleitner, C. Kerl, The potential of low-code development in the manufacturing industry, in: 2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA), IEEE, Sinaia, Romania, 2023, p. 1–8. URL: https://ieeexplore.ieee.org/document/10275503/. doi:10.1109/ETFA54631.2023.10275503.

[5] D. Gentner, A. L. Stevens, Mental models, Psychology Press, Hove, East Sussex, England, 2014.