LLM-based Iterative Requirements Refinement in FSM with IEC 61499 Code Generation

Valeriy Vyatkin*‡§, Sandeep Patil‡§, Dmitrii Drozdov§, Anatoly Shalyto†
*Department of Electrical Engineering and Automation, Aalto University, Espoo, Finland
‡Department of Computer Science, Electrical and Space Engineering, Luleå University of Technology, Luleå, Sweden
§Flexbridge AB, Luleå, Sweden
†Independent researcher
Emails: vyatkin@ieee.org, sp@flexbridge.se, dd@flexbridge.se, anatoly.shalyto@gmail.com

***Abstract*—This paper presents the Function Block Assistant (fbAssistant), an LLM-backed tool prototype for developing control logic in industrial automation. fbAssistant interprets natural language requirements and automatically generates state machines and their function blocks implementation. The study demonstrates iterative refinement, simulation validation, and deployment using EcoStruxure Automation Expert. The proposed approach aims at improved efficiency and accuracy in the development of automation software.**

***Index Terms*—Requirements engineering, finite-state machines, IEC 61499, LLM**

I. INTRODUCTION

Industrial automation software development is a very resource-consuming process. It requires the creation of precise control logic to ensure reliable and safe operation. Traditional development methods require extensive experience and manual programming. The development of automation systems starts with requirements written in natural language and is accompanied by various auxiliary documents, such as I/O assignment tables, piping and instrumentation diagrams (P&ID), and other technical drawings and wiring diagrams.

State machines are a fundamental concept in digital design, providing a structured way to represent sequential logic and control flows. They allow systems to transition between well-defined states based on input conditions, ensuring predictable and deterministic behaviour. In digital circuits design, state machines are widely used in the design of finite control logic, sequential processors, and embedded systems, which makes them essential for automation and control applications.

The concept of direct state-machine programming, or more generally, programming using visual models, such as state machines or flow-charts, has been advocated by the authors in many previous publications, including [1]–[4].

In industrial automation, state machines play a crucial role in structuring control logic for complex processes. They enable engineers to model different operational states of machines and define transitions based on specific inputs, such as sensor readings or user commands. This structured approach helps ensure safety, reliability, and efficiency, as each state is explicitly defined and unexpected behaviour is minimised. Furthermore, state machines simplify troubleshooting and validation, as engineers can systematically analyse transitions and conditions that lead to a particular operational state.

A key advantage of using state machines in industrial automation is their ability to facilitate the step-by-step refinement of natural language (NL) requirements. Engineers typically start by defining automation goals in informal terms, which are then incrementally transformed into structured states and transitions. This process allows for continuous refinement, enabling the incorporation of additional safety features, optimisation constraints, and operational refinements. Using AI-driven tools, such as the Function Block Assistant, engineers can automate this transformation, improving the speed and accuracy of control logic development.

The approach and the Function Block Assistant (fbAssistant) proof-of-concept tool, created by Flexbridge AB [5], leverage Large Language Models (LLM) to generate automation logic from natural language descriptions, expediting development while ensuring compliance with formal models.

Research on retrieval-augmented generation (RAG) methods [6] has shown that combining pre-trained models with external knowledge sources enhances the reliability of AI-generated output. This aligns with the Function Block Assistant's approach, where structured knowledge from industrial control systems is integrated with AI reasoning to produce verifiable and safe automation logic.

The rest of the paper is structured as follows. Section II puts this work in the context of generative AI progress. Section III presents the proposed methodology of specifications refinement using an example of pneumatic cylinder. Section IV adds details of another, more complicated case study of a pick and place manipulator. Section V summarises the contribution with discussion of results. Section VI concludes the work with outlook and future work plans.

II. RELATED WORK: AI GRAND CHALLENGES AND SOFTWARE ENGINEERING

The intersection of artificial intelligence and software engineering presents several grand challenges that could lead to revolutionary advancements in both fields. AI has demonstrated impressive capabilities in natural language processing, decision making, and automated reasoning, but its application to software and systems engineering is still in an evolving phase.

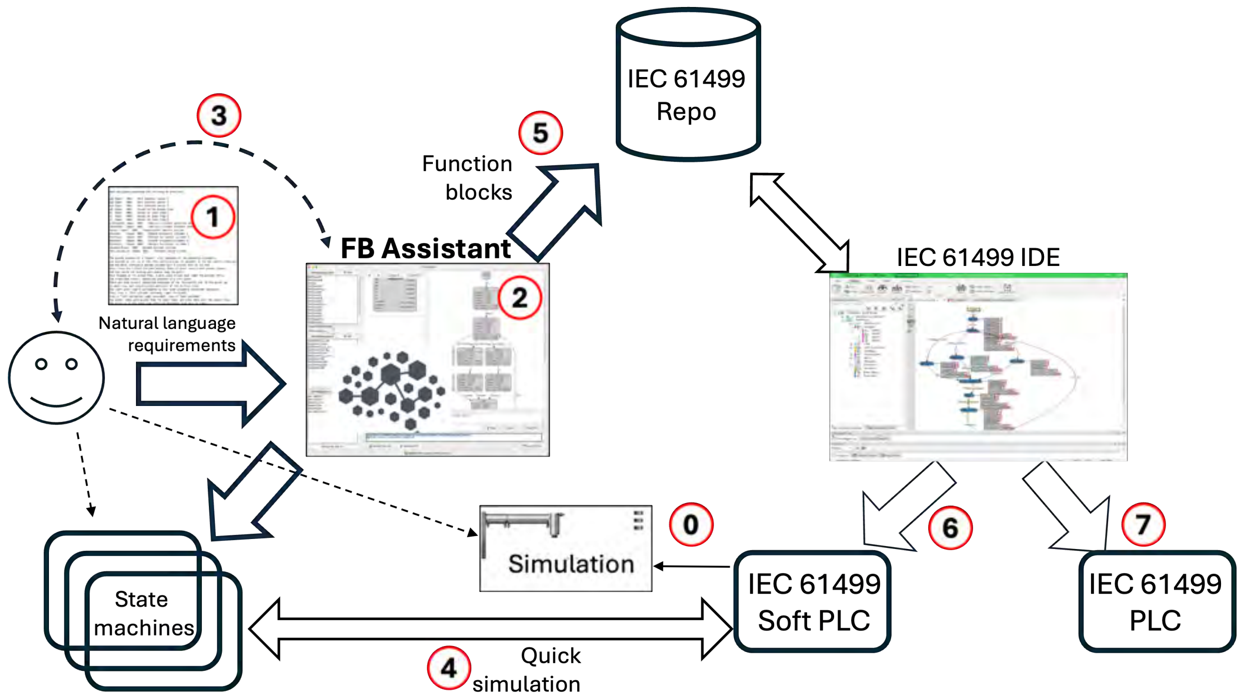

Fig. 1: Overview of the Function Block Assistant workflow, showing the step-by-step process from natural language requirements to executable control logic.

One of the main challenges in AI is the development of generalisable reasoning systems that can model complex hierarchical decision-making processes [7]. In software engineering, this challenge translates into AI's ability to understand, refine, and generate structured programme representations from informal specifications. A key area of exploration is neurosymbolic AI, which combines statistical learning with formal logical reasoning [8]. The Function Block Assistant aligns with this trend by integrating symbolic state machine representations with AI-driven refinement processes, bridging the gap between human intent and executable logic.

Another challenge is to develop AI systems that can facilitate requirement refinement in digital design, an essential aspect of the engineering of complex automated systems. AI models need to go beyond simple pattern matching and incorporate mechanisms reminiscent of human cognitive processes, such as abstraction, hierarchical reasoning, and error correction [9]. The study of human cognition suggests that conceptual refinement occurs through iterative adjustments, where humans learn by interacting with an environment and refining mental models [10]. Similarly, the Function Block Assistant iteratively refines its generated control logic based on user feedback, reflecting a fundamental aspect of human cognitive modelling.

From the software engineering perspective, increasing the complexity of the system requires new AI models that can handle higher levels of abstraction, composability, and explainability. Current AI-driven software generation techniques often rely on neural networks that lack intrinsic explainability, making it difficult for engineers to verify their correctness [11]. To address this, AI systems must incorporate structured reasoning frameworks that allow human-interpretable justifications of their decisions. Recent research on retrieval-augmented generation [6] suggests that combining external knowledge sources with AI reasoning improves system transparency, which is crucial for AI-assisted software engineering.

Furthermore, AI-driven automation introduces new challenges related to verification and validation. Traditional software verification techniques involve formal methods and model checking, but scaling these approaches to AI-generated code remains a fundamental challenge. The integration of formal verification with AI-generated control logic is essential to ensure safety and reliability, especially in industrial automation settings, where failures can have significant consequences.

Finally, AI's influence on software engineering is also bidirectional: progress in software and systems engineering can drive new advancements in AI. As AI becomes increasingly integrated into engineering workflows, there is a growing need for AI models that can reason about complex software architectures, perform automated debugging, and self-correct erroneous logic. These capabilities require a deeper understanding of the cognitive processes involved in software development, suggesting that future AI models must incorporate meta-learning strategies inspired by human problem solving techniques.

Recent work on integrating large language models (LLMs) into industrial automation has demonstrated their ability to improve intelligent analysis and visualisation of IEC 61499

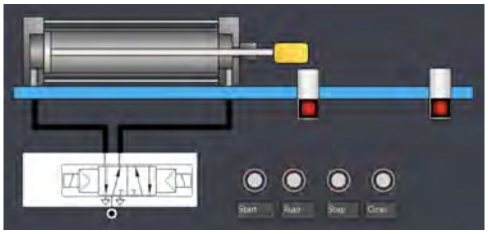

Fig. 2: Pneumatic cylinder system.

control systems [12]. Using AI-powered reasoning, these approaches improve system transparency and facilitate more efficient debugging and optimisation of automation workflows.

Generative AI has also been explored as a co-pilot for rapid prototyping of IEC 61499 control applications, significantly reducing development time and increasing automation efficiency [13]. These advancements align with the Function Block Assistant's goal of bridging the gap between human intent and executable control logic by automating function block generation and refinement.

III. Methodology

The Function Block Assistant (fbAssistant) follows an iterative methodology to develop automation logic. The workflow of the tool is presented in Fig. 1.

We illustrate the application of the Function Block Assistant using a simple example of a pneumatic cylinder. This system, shown in Fig. 2, consists of a double-acting pneumatic cylinder controlled by two actuating signals that move forward and backward and are monitored by two position sensors that indicate the left and right positions.

The process begins (Step (1) in Fig. 1) with the user providing natural-language descriptions of the required control logic. These descriptions are then processed by an LLM, which generates a corresponding state machine representation.

For best efficiency of requirements refinement, it is recommended that fbAssistant is run in parallel with the IEC 61499 development environment, such as EcoStruxure Automation Expert (EAE) [14], in which a closed-loop simulation model of the system is already running, for example, in case of the cylinder, as the one shown in Fig. 3. The function block application follows the Cyber-Physical Components architecture [15] - the adaptation of object-oriented Model-View-Controller pattern for component-based automation.

The overall workflow is briefly outlined as follows:

1) Closed-loop simulation environment is created in IDE with empty controller function block connected in the loop to function block implementing simulation model of the process, and function block(s) implementing interactive HMI elements, such as buttons and indicators.
2) Natural language requirements are prepared and input to the tool;
3) State machine is created by LLM and displayed for visual inspection;
4) State machine is refined in several iterative steps;

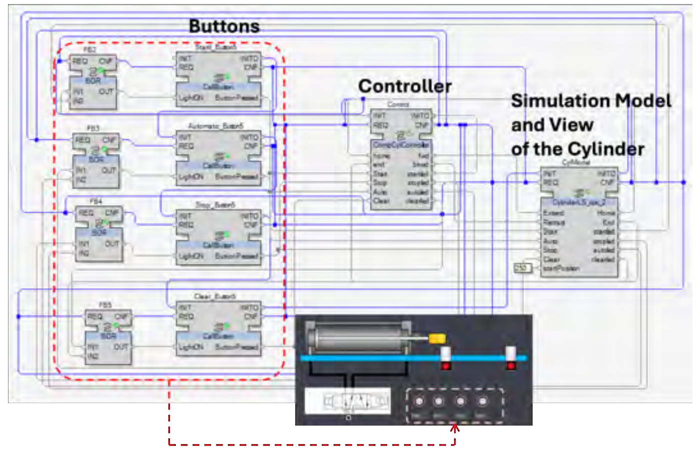

Fig. 3: Closed-loop HMI-Controller-Model environment in IEC 61499 for the Cylinder.

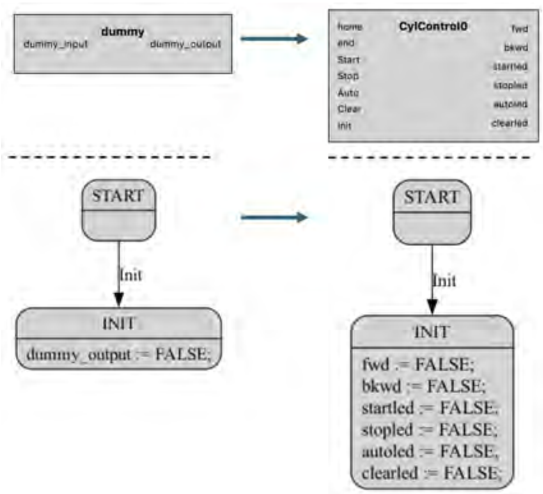

Fig. 4: Initial dummy state machine module transformed to a state machine with proper interface based on IO list and initialisation of output signals.

5) The correctness of the logic is checked by simulation in the loop with model running under IDE;
6) Equivalent function block is generated and "implanted" to the project in IDE;
7) The updated closed-loop environment is deployed to softPLC runtime and tested with the autogenerated controller FB;
8) Deployment to the real PLC.

In fbAssistant, the development starts with an empty state machine module in Figure 4 (left side). Application of the initial prompt, describing the interface of the system, forms the interface of the module, and initialises all outputs to FALSE Fig. 4 (right side).

The initial user prompt to fbAssistant included the following instructions:

The pneumatic double-acting cylinder system is described by the following input and output interface. Add these signals to the module's interface and remove dummy signals.
In addition add Init input for initialisation. Do not modify the state machine.

The interface specification is based on the IO list of the system:

home	Input	BOOL	Home position sensor (detects if the cylinder is in the home position)
end	Input	BOOL	End position sensor (detects if the cylinder is fully extended)
Start	Input	BOOL	Start button for the cylinder
Stop	Input	BOOL	Stop button to halt cylinder operation
Auto	Input	BOOL	Auto mode selector switch
Clear	Input	BOOL	Clear/reset button
fwd	Output	BOOL	Extend (move cylinder forward)
bkwd	Output	BOOL	Retract (move cylinder backward)
startled	Output	BOOL	Indicator light for the start signal
stopled	Output	BOOL	Indicator light for the stop signal
autoled	Output	BOOL	Indicator light for the auto mode signal
clearled	Output	BOOL	Indicator light for the clear/reset signal
Init	Input	BOOL	Initialization signal

The next step is to give the assistant essential information about the desired behaviour of the system and provide some expectations about the structure of the state-machine.

When the system starts up, it initializes and makes sure the cylinder is in a known state. If the cylinder is not at home, the controller automatically moves it backward until it reaches the home position. Once at home, the cylinder waits for a user command. When the user presses "Start", the cylinder extends forward until it reaches the end position. If "Start" is pressed again, the cylinder moves backward to home. In addition to the existing states START and INIT, the resulting state machine must have states: TOLEFT, ATLEFT, ATRIGHT, GO and GOBACK. In TOLEFT cylinder moves to the home position state ATLEFT after initialisation. The condition transition from INIT to TOLEFT should be Always, i.e. always true. In the state ATLEFT the cylinder is standing still and waiting for the Start button. Another static state is ATRIGHT, where cylinder should wait for the Start button before moving back to the home position. Two motion states GO and GOBACK implement the cylinder's motion in the forward and reverse directions. Make sure to always reset the control signals when the motion should end.

Based on these requirements, fbAssistant has generated the state machine in Fig.5, left side.

Once the state machine is created, it is visualised for user verification. This visualisation helps identify any missing transitions or incorrect states. If adjustments are needed, the user can submit requests for corrections also in natural language form. Once the user feels that the state-machine is correct, they can immediately run it using the built-in interpreter in fbAssistant. The interpreter connects to the simulation model running in the EAE environment thus enabling simulation in the loop. For example, for the cylinder case, the simulator has exactly the interface shown in Fig.2, so the user can press the start button and make sure the state machine works properly by observing the cylinder's motion.

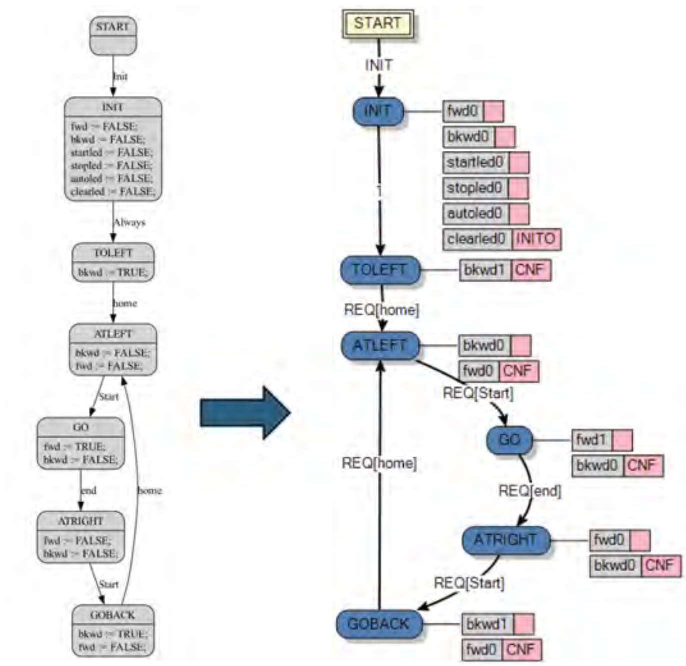

Fig. 5: Auto-generated state machine (left), and ECC of the function block autogenerated from the state machine (right).

After initial validation of the state machine, an equivalent IEC 61499 function block can be generated so that it could be immediately open in an industrial automation tools such as EAE. Execution control chart of the generated function block is shown in Fig.5, right side. The block can be seamlessly put in the closed-loop environment from Fig.3 by changing the function block type of the controller to that of the newly generated function block. Then the control logic is deployed to a virtual or physical environment for testing as a part of the closed-loop application.

Testing involves simulating the generated control logic within a virtual commissioning environment. This allows engineers to verify the expected behaviour before deploying it to the actual hardware. The refinement process ensures that any discrepancies between expected and actual behaviour are resolved prior to full implementation.

As an intermediate summary, the state machine was generated based on natural language descriptions of the required behaviour. The AI-assisted development included the following steps:

1) The initialisation logic was added to move the cylinder

to the left position on start-up.

2) The requirements were extended to describe HMI logic, first starting with the START button, then, adding the light behind the button and describing its behaviour.

Once validated, the function block was generated and deployed to the EcoStruxure Automation Expert system. The final version ensured smooth and reliable cylinder operation, with correct state transitions and improved safety handling.

In one of the following state machine refinement steps we added the emergency stop button functionality as illustrated in Fig.6.

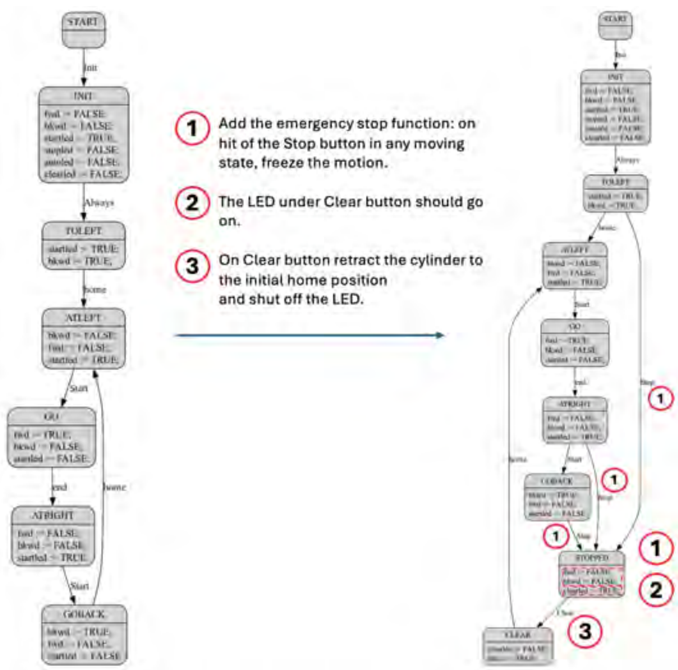

Fig. 6: Adding emergency stop button functionality.

The prompt for this requirement can be divided onto three parts as shown in the figure. It describes behaviour of two buttons Stop and Clear in process of implementing the emergency stop and clearing it. The diagram shows the places in the modified state machine which are related to the corresponding parts of the prompt. The prompt was implemented successfully from the first run.

IV. Pick and Place Manipulator Use Case

Another use-case of the Function Block Assistant is the development of control logic for pick and place manipulator.

These robotic systems require precise sequencing of operations, including gripping, lifting, transporting, and releasing objects. Using fbAssistant, engineers can define the high-level behaviour of the manipulator in natural language, allowing the AI to generate an initial state machine.

The pick-and-place manipulator Fig. 7, used in this experiment handles a total of 4 locations where items either appear or disappear at: 3 input trays and 1 output tray. The objective of the pick-and-place machine is to transfer these items, referred to as workpieces, from the input trays to the output tray. The input trays can hold one workpiece each at a time, and the output tray disposes of the workpieces as they are placed in it.

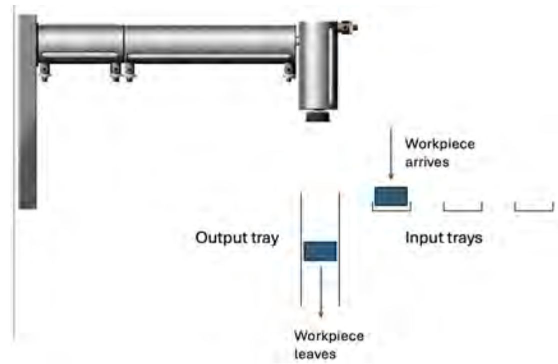

Fig. 7: Pick and place manipulator.

As in the previously considered Cylinder case, the development process began with defining the system's inputs and outputs, such as actuator control signals and position sensors. The following description of the desired behaviour was provided:

> *The initial position of the robot is when all cylinders are retracted. Desired behaviour: If workpiece is available in one of input trays, the robot must extend the required number of horisontal cylinders to reach the tray, then extend the vertical cylinder, switch the vacuum on, wait for suction sensor confirms the workpiece is sucked, then lift the vertical cylinder, retract horisontal cylinders, retract back to the output tray, and if the output tray is empty - extend the vertical cylinder, drop the workpiece, and go to the initial position.*

The AI generated a state machine with initial logic, which was iteratively refined based on simulation feedback. The assistant ensured that transitions between states, such as gripping and releasing, were executed in the correct sequence and prevented undesired simultaneous movements.

The iterative dialogue between the developer and the AI led to enhancements such as:

- Preventing the manipulator from dropping objects prematurely.
- Ensuring a safe return to the home position.
- Optimizing motion sequences to reduce cycle time.

Figure 8 illustrates one step in the state machine refinement process supported by fbAssistant. Given the incomplete state machine of PnP controller on the left, the user provided five comments in natural language, each of which was correctly addressed by the assistant, making the corresponding modifications to the state machine.

The iterative refinement process has converged to the state machine shown in Fig. 9.

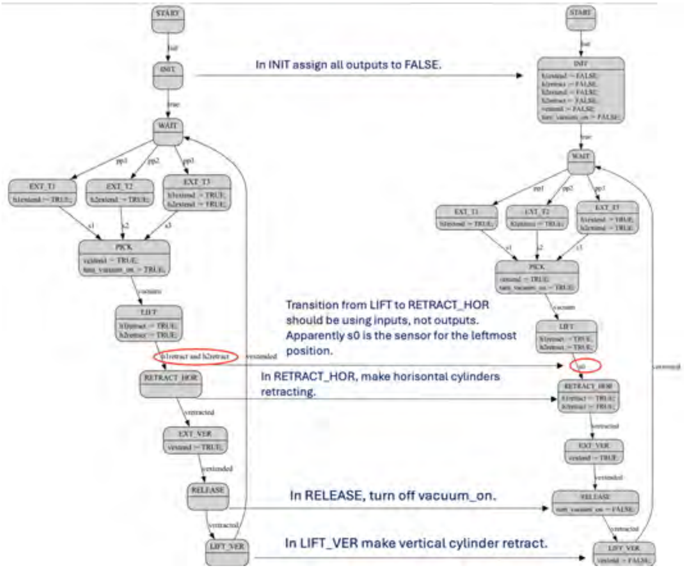

Fig. 8: A step in the refinement of PnP controller state machine.

Once the refined state machine was validated in simulation, it was converted into a function block and deployed within EcoStruxure Automation Expert. The real-world testing confirmed that the AI-generated control logic met the expected functional requirements.

V. Discussion

The initial experiments with the proof-of-concept tool implementing the AI-driven approach prove significant reduction of development time while maintaining high accuracy.

The iterative refinement process with fbAssistant helps developer to better understand their intentions formulated in natural language. Sometimes LLM is quite sharp in handling misinterpretations or imprecisions, which otherwise could lead to ambiguities.

However, minor manual adjustments were required to fine-tune state transitions and UI interactions.

Examples of famous "hallucinations" include creation of states without predecessor states, or cleaning states from actions on request to remove only one action, which proves the need for more explicit prompt commands.

On the other side, fbAssistant has shown stable capability in handling natural language requirements, even in different languages. For example, we repeated the Cylinder use case with requirements presented in Mandarine with the same result.

VI. Conclusions and Outlook

fbAssistant demonstrates the feasibility of AI-assisted control logic generation for industrial automation. The iterative approach allows engineers to refine automation logic efficiently, improving overall system safety and performance.

Future work plans are multi-fold: One direction of work will focus on improving AI understanding of ambiguous requirements and integrating additional safety constraints as reported in the paper [16]. The available formal verification framework described in [17] is planned to be used for formal verification of the autogenerated function blocks, while the automated testing from [18] can be used for their extensive testing.

Another direction of work relates to application use case of gradual change of the manufacturing facility to produce a new product on it, by applying iteratively changing requirements.

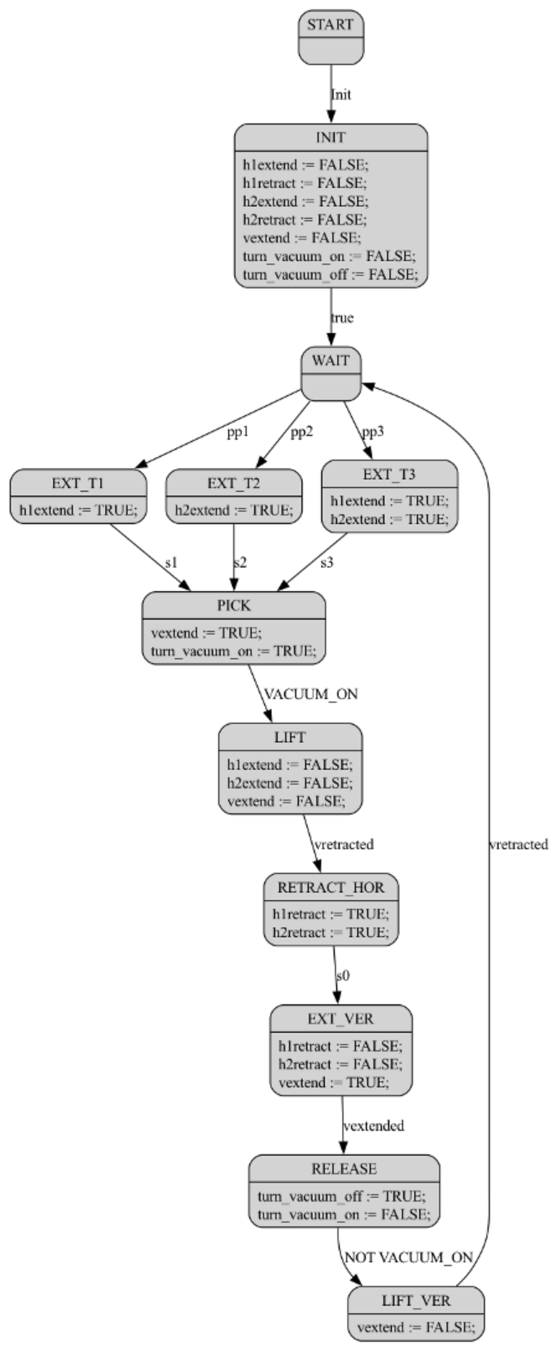

Fig. 9: Final result of PnP control development.

VII. Acknowledgements

This work was supported, in part, by the Horison Europe project MEDUSA which is focussing on the development of manufacturing as a service platform supported by AI.

References

[1] L. Naumov and A. Shalyto, "Automata theory for multi-agent systems implementation," in *IEMC'03 Proceedings. Managing Technologically Driven Organizations: The Human Side of Innovation and Change (IEEE Cat. No. 03CH37502)*. IEEE, 2003, pp. 65–70.

[2] B. Yartsev, G. Korneev, A. Shalyto, and V. Kotov, "Automata-based programming of the reactive multi-agent control systems," in *International Conference on Integration of Knowledge Intensive Multi-Agent Systems, 2005.* IEEE, 2005, pp. 449–453.

[3] A. Shalyto, L. Naumov, and G. Korneev, "Methods of object-oriented reactive agents implementation on the basis of finite automata," in *International Conference on Integration of Knowledge Intensive Multi-Agent Systems, 2005.* IEEE, 2005, pp. 460–465.

[4] V. Vyatkin, G. Ivanov, V. Fedorenko, and V. Osheter, "Chartware - an approach to structured visual programming for industrial logic control," *Controllers and Control Systems*, vol. 3, no. 4, pp. 33–44, 1996.

[5] (2025) Function block assistant. Universal Automation. [Online]. Available: https://www.youtube.com/live/aR20KBmZnA4?si=wxyMOcAX4tirRgQf

[6] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. tau Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 9459–9474. [Online]. Available: https://arxiv.org/abs/2005.11401

[7] R. Caruana, Y. Lou, J. Gehrke, P. Koch, M. Sturm, and N. Elhadad, "Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission," in *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015. [Online]. Available: https://people.dbmi.columbia.edu/noemie/papers/15kdd.pdf

[8] L. Guan, K. Valmeekam, S. Sreedharan, and S. Kambhampati, "Leveraging pre-trained large language models to construct and utilize world models for model-based task planning," in *Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS)*, 2023. [Online]. Available: https://arxiv.org/abs/2305.14909

[9] C. Hu, Y. Hu, H. Cao, T. Xiao, and J. Zhu, "Teaching language models to self-improve by learning from language feedback," in *Findings of the Association for Computational Linguistics (ACL)*, 2024. [Online]. Available: https://aclanthology.org/2024.findings-acl.364/

[10] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, 2006, pp. 1735–1742.

[11] J. Backes, P. Bolignano, B. Cook, C. Dodge, A. Gacek, K. Luckow, N. Rungta, O. Tkachuk, and C. Varming, "Semantic-based automated reasoning for aws access policies using smt," in *Formal Methods in Computer-Aided Design (FMCAD)*, 2018, pp. 1–9.

[12] M. Xavier, T. Laikh, S. Patil, and V. Vyatkin, "LLM-powered multi-actor system for intelligent analysis and visualization of iec 61499 control systems," in *IECON 2024-50th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2024, pp. 1–8.

[13] P. Ovsiannikova, T. Liakh, P. Jhunjhunwala, and V. Vyatkin, "Generative ai co-pilot for rapid prototyping of iec 61499 control applications," in *2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2024, pp. 1–4.

[14] "Schneider Electric EcoStruxure Automation Expert," https://www.se.com/us/en/product-range/23643079-ecostruxure-automation-expert/, 2023, accessed: February 11, 2025.

[15] G. Zhabelova, C.-W. Yang, S. Patil, C. Pang, J. Yan, A. Shalyto, and V. Vyatkin, "Cyber-physical components for heterogeneous modelling, validation and implementation of smart grid intelligence," in *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*. IEEE, 2014, pp. 411–417.

[16] A. King and V. Vyatkin, "LLM-based iterative refinement of finite-state machines with STPA controller constraints and generation of IEC 61499 code," in *Proceedings of the 2025 IEEE 30th International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2025.

[17] G. Lilli, M. Xavier, E. Le Priol, V. Perret, T. Liakh, R. Oboe, and V. Vyatkin, "Formal verification of the control software of a radioactive material remotehandling system based on IEC 61499," *IEEE Open Journal of Industrial Electronics Society*, 2023.

[18] V. Vyatkin and R. Rumiantcev, "Framework for faster-than-real-time testing of IEC 61499 applications with embedded process simulation," in *2024 IEEE 22nd International Conference on Industrial Informatics (INDIN)*. IEEE, 2024, pp. 1–5.