ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP001

WORKING TOGETHER FOR SAFER SYSTEMS: A COLLABORATION MODEL FOR VERIFICATION OF PLC CODE

Ignacio D. Lopez-Miguel*, TU Wien, Vienna, Austria,
Borja Fernández Adiego†, Enrique Blanco Viñuela, CERN, Geneva, Switzerland,
Matias Salinas, Christine Betz, GSI, Darmstadt, Germany

Abstract

Formal verification techniques are widely used in critical industries to minimize software flaws. However, despite the benefits and recommendations of the functional safety standards, such as IEC 61508 and IEC 61511, formal verification is not yet a common practice in the process industry and large scientific installations. This is mainly due to its complexity and the need for formal methods experts. At CERN, the PLCverif tool was developed to verify PLC programs formally. Although PLCverif hides most of the complexity of using formal methods and removes barriers to formally verifying PLC programs, engineers trying to verify their developments still encounter different obstacles. These challenges include the formalization of program specifications or the creation of formal models. This paper discusses how to overcome these obstacles by proposing a collaboration model that effectively allows the verification of critical PLC programs and promotes knowledge transfer between organizations. By providing a simpler and more accessible way to carry out formal verification, tools like PLCverif can play a crucial role in achieving this goal. The collaboration model splits the specification, development, and verification tasks between organizations. This approach is illustrated through a case study between GSI and CERN.

INTRODUCTION

Programmable Logic Controllers (PLCs) find extensive utilization in industrial automation, encompassing safety and standard control systems not only at establishments like CERN and GSI but also within diverse process industries and other scientific installations. The incorrect behavior of the PLC programs can yield considerable repercussions, such as property damage, environmental harm, or, in certain instances, even personal injuries. Consequently, the assurance of their accurate functionality holds paramount importance. Although testing has long been the conventional means of validating PLC programs, its effectiveness often falls short as the sole verification method. Even when automated, testing cannot achieve exhaustive coverage and thus lacks the ability to guarantee the absolute correctness of a given logic. Certain categories of requirements, such as safety specifications (which demand the prevention of unsafe states) or invariants (formulas that must hold throughout all conceivable system runs), can pose substantial challenges and might even be unfeasible to assess through testing alone.

Model checking is a formal verification technique that complements the testing activities to fully validate and verify PLC programs. It involves evaluating the fulfillment of formalized requirements upon a mathematical model of the system under scrutiny. This assessment encompasses every feasible combination of inputs and all potential execution paths. Additionally, if a violation is found, the technique provides the path leading to the breached requirement.

The broad adoption of model checking in the realm of PLCs encounters a two-fold challenge: Firstly, creating the mathematical model that represents the system being analyzed can require an in-depth understanding of the model-checking tools. Secondly, a multitude of real-life PLC logics are characterized by their complexity, leading to what is known as state-space explosion problem, that is, a large number of potential input combinations and execution paths that exceed the bounds of exhaustive exploration.

In September 2020, CERN released the PLCverif platform under an open-source license, aiming to facilitate the utilization of model-checking tools among PLC developers. This goal was achieved by automating the conversion of PLC programs into their corresponding mathematical models. Subsequently, a series of abstraction algorithms were integrated to address the state-space explosion challenge.

However, the incorporation of model-checking tools into the design and development process is far from simple, encountering various challenges. Large projects involve distinct teams that must engage in clear and unambiguous communication. Thus, achieving this integration is not trivial, particularly when the initial specifications are not formalized and lack precision. These specifications are usually written using natural language. Consequently, when the PLC developer implements the specifications, they must interpret these instructions, potentially resulting in a PLC program that deviates from the initially envisioned design conceived by the team responsible for creating the specifications.

Through collaborative efforts, by exchanging expertise and tools, constructing safe systems can become more viable. Such a collaboration took place between CERN and GSI, wherein CERN helped GSI to formalize requirements and to formally verify their PLC code.

Upon termination of the collaboration, a number of discrepancies between the specification and the PLC code were found. This discovery enhanced the understanding of safety engineers and PLC developers at GSI about the functioning of their PLC programs. It enabled them to refine the specification, rectify errors, and improve the system overall. Moreover, this collaboration yielded advantages for CERN

* ignacio.lopez@tuwien.ac.at
† borja.fernandez.adiego@cern.ch

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP001

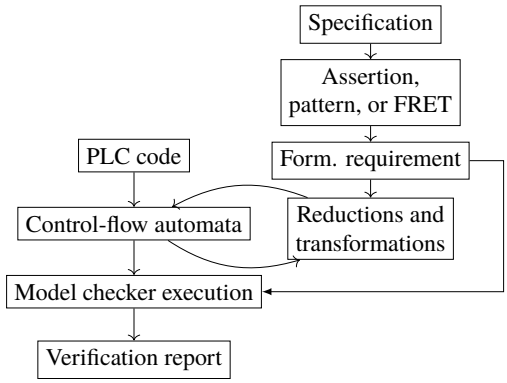

Figure 1: Workflow of the verification process utilizing PLCverif.

since it led to enhancements in the PLCverif tool.

The aim of this paper is to introduce this innovative collaboration and its results, with the intent of inspiring other institutions or enterprises to embark on a similar trajectory and build safer systems.

The rest of the paper will present an overview of the PLCverif framework (section *PLCverif*), a description of the Collaboration model developed (section *Collaboration model*), a specific GSI case study presenting the work and results (section *GSI case study*), and finally a Conclusion summarising improvements that emerged from this work and an outlook to the future (section *Conclusion*).

PLCVERIF

PLCverif[1] [1, 2] stands as a flexible and expandable framework born at CERN. Its purpose is to facilitate the formal verification of PLC programs. The framework is designed to be customizable and is built upon a plugin architecture.

The verification process using PLCverif is depicted in Figure 1 [2, 3]. The process begins with the transformation of the initial specification into a set of formal requirements. There are three potential pathways for this formalization: (i) representing the requirements as *assertions*, which will then be directly embedded into the PLC code, (ii) integrating the necessary variables and conditions into *patterns* (pre-defined templates), or (iii) utilizing *FRET* to express the requirement in a structured natural language, a recent integration of the NASA Formal Requirements Elicitation Tool (FRET)[2] [4, 5].

Subsequently, both the PLC code and the formalized requirements undergo translation, creating an intermediate model. This model is represented as a collection of control-flow automata (CFA) [6]. Certain reductions and transformations are performed to simplify the model. The next phase involves the execution of a model checker, an external tool that generates a counterexample if the property is violated, i.e., a combination of the input variables that leads to a violation of the property. The verdict of the verification and the counterexample (in case of violation) are presented to the user in an intuitive and comprehensible verification report.

PLCverif has been used successfully in the verification of various complex safety-critical systems [7–9].

[1] PLCverif is available on `https://gitlab.com/plcverif-oss`.
[2] FRET is available on `https://github.com/NASA-SW-VnV/fret`.

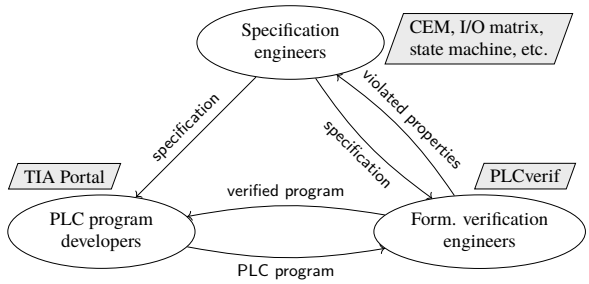

Figure 2: Diagram depicting the different roles of the collaboration, which information is shared, and which tools are used.

COLLABORATION MODEL

This section describes the proposed collaboration model, including the roles of the collaboration members, examples of how specifications of safety-critical PLC programs should be written according to the functional safety standards, and the kind of results that formal verification can bring to the project.

Roles

Figure 2 depicts the proposed collaboration model. It is composed of the following roles:

- Specification engineers. They comprise control and safety experts with extensive knowledge of systems and processes. Due to this expertise, they are responsible for analyzing the systems and writing their technical specifications. They can use different formalisms or tools to specify requirements, as detailed in subsection *Specification*. These methods include cause-and-effect matrices, I/O matrices, and state machines, allowing them to define and document the project's specifications unambiguously.

- PLC program developers. They receive the requirements from the specification engineers and implement them in a PLC program. The code they produce should be modular, maintainable, and well-documented.

- Formal verification engineers. Given the specification and the PLC code, they verify that the latter behaves exactly as written in the specification. This verification is done with PLCverif. If there are discrepancies, they suggest fixes and help to improve the specification and the understanding of the behavior of the PLC program.

There are many reasons to structure this collaboration in this way. It comes from the typical scenario where separate teams handle specification and code creation. Therefore, a clear distinction between these two entities is essential.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP001

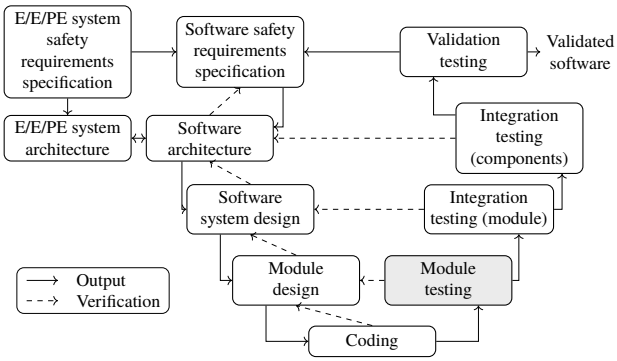

Figure 3: Software systematic capability and the development life cycle (the V-model), cf. [10].

While advancements like PLCverif hold the potential to bridge this gap in the future, our experience shows that PLC developers do not use formal verification in their PLC program development process yet. This, together with technical barriers and lack of background in formal verification, implies that the most optimal way to use formal verification is to have an external team that takes on this role.

The formal verification team should try to involve the other counterparts as much as possible to ensure their efforts are precisely aligned. This collaborative dynamic facilitates the exchange of expertise, leading to a point where, by the collaboration's culmination, internal personnel can independently execute verification tasks without external guidance.

The final deliverable at the end of the collaboration should yield a set of reports detailing the properties violated by the given code and suggestions for remedies. This is an iterative process, in which formal verification engineers will reevaluate the amended specification and code, reinitiating the verification process until all errors are successfully addressed. This refinement would lead to an error-free verification.

This collaboration is fully compliant with the well-known V-model for software development. The IEC 61508-3 functional safety standard [10] recommends the use of the V-model when describing the software life-cycle requirements for safety systems (Figure 3). It is a framework used in software engineering to guide the specification and validation processes. It illustrates the relationship between different phases of software specification (left side) and their corresponding validation phases (right side).

The V-model promotes testing alignment with the corresponding specification activities, leading to a well-structured and well-tested software product. It emphasizes the importance of testing at each stage of development, aiming to catch defects early in the process and reduce the likelihood of major issues arising in later stages.

This aligns precisely with the central focus of verification within software development. Its purpose is the identification of deficiencies at the early stages of the development process, facilitating rapid resolution. This helps to perform a much faster testing phase, yielding fewer errors. It is important to note that verification is not intended to replace testing, but rather to serve as a valuable complement.

Table 1: Example of a CEM representing when the output signal Out_1 has to be true. A disjunctive expression is created from the two clauses A1 and A2. A1 is a conjunction, and A2 is a conjunction whose elements are negated.

		Outputs	
		Out_1	Out_2
Inputs	In_1	A1	A1
	In_2	A1	A1
	In_3	NA2	A1
	In_4	NA2	A1

Table 2: Example of an I/O matrix representing when the signals Out_1 and Out_2 shall be set or reset according to the inputs. The specification can be ambiguous if the inputs are not mutually exclusive or their priority is not defined.

		Outputs	
		Out_1	Out_2
Inputs	In_1	Reset	Reset
	In_2	Set	Reset
	In_3	Set	Set

In Figure 3, the verification activities undertaken during the collaboration can be situated within the realm of *module testing*. This occurs at the earliest possible stage after code development. Identified problems in the specification or the code at this point can be easily amended.

In some situations, errors appear during operation, incurring substantial costs or even severe accidents and necessitating the reiteration of numerous steps. These types of errors can often be detected by verification in an early phase of the development process, as it excels in uncovering problems arising from corner cases. Thus, verification can not only help to develop safer code, but it can also profoundly influence cost reduction in the development process.

Specification

Diverse formal representations were used to represent different requirements. These representations must possess qualities of simplicity, clarity, and conciseness. This is imperative to effectively capture the requirements and streamline the processes of designing, implementing, and verifying PLC programs.

Some of the possible formalisms recommended by the functional safety standards are listed below.

- A *Cause and Effect Matrix (CEM)* [11] is a concise and intuitive graphical depiction of Boolean expressions. It is particularly suitable for stateless logic, where outputs are solely influenced by combinations of current input signals. CEM is widely embraced for specifying interlock logic. Variants of CEMs exist, and companies often adopt the semantics that align with their processes and engineering approaches. Some PLC manufacturers, such as Siemens, have incorporated CEM into their engineering tools, exemplified by the SIMATIC Safety

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP001

Matrix. The exemplary CEM from Table 1 represents in a compact way the conditions to make the output variables Out_1 and Out_2 true. Its equivalent logic formulas are

$$Out\_1 = (In\_1 \wedge In\_2) \vee (\neg In\_3 \wedge \neg In\_4),$$
$$Out\_2 = \bigwedge_{i=1}^{4} In\_i.$$

- An *input-output matrix* (I/O matrix) depicts the relationship between input variables and output variables in a systematic way. It can be used to represent the transformation applied to input signals to produce output signals, which is particularly useful when dealing with systems that process multiple signals simultaneously. With this formalism, one needs to be sure that the inputs are mutually exclusive or to specify their priorities. The I/O matrix from Table 2 gives the conditions to set or reset the output signals Out_1 and Out_2 according to some given inputs.
- A *state machine* is a graphical representation used to depict the behavior of a system that undergoes distinct states and transitions between them. It consists of states representing the various conditions or phases the system can be in and transitions illustrating how the system moves from one state to another based on certain conditions. They provide a clear overview of system behavior and help design, analyze, and document complex processes or systems. Figure 4 shows a simple state machine that changes from two modes depending on the requests. For a real example, one can refer to Figure 4.4 from [12].
- A *logic diagram* is a visual and formal representation of logical relationships and conditions. It represents an electronic circuit designed to perform logical operations based on input signals and produce output signals according to predetermined logic rules. It utilizes various logic gates, which are basic building blocks that manipulate Boolean signals according to logical operations such as AND, OR, NOT, and XOR. It is especially useful for requirements that involve conditions, constraints, or logical dependencies. At CERN, grassedit[3] is an ongoing project that facilitates the creation and simulation of logic diagrams. The tool also automatically generates assertions to insert in PLCverif directly from grassedit diagrams. An example of a logic diagram is depicted in Figure 5, where the following formula is represented: $Out\_1 = (In\_1 \vee In\_2) \wedge In\_3$.
- An *assertion* is a condition used to express the expected behavior or a specific property of a program at a particular point in the code's execution. Although this is typically used during software development, it can be used to formalize a given requirement. They are particularly helpful in expressing safety properties, i.e., a state can never be reached. They are then easily transferred into PLCverif. Assertion 1 has the same meaning as the first row of the I/O matrix of Table 2.

[3] Grassedit is available on `https://grassedit.web.cern.ch/`.

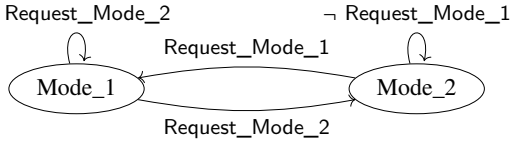

Figure 4: Example of a state machine that indicates how a system transitions from one mode to another.

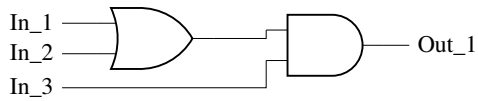

Figure 5: Logic diagram, specifying when the output signal Out_1 has to be true.

Assertion 1 *If* In_1 *is true, then* Out_1 *shall be reset, and* Out_2 *shall be reset.*

Furthermore, CERN made an effort to formally specify PLC programs, resulting in PLCspecif [13]. It combines different formalisms, such as state machines, I/O matrices, and invariants (assertions).

The IEC61511-2 standard [14] provides guidelines and examples for applying this IEC61511-1 Clauses. Part 2 references the specification methods CEM, state machines, and logic diagrams. For instance, Annex F (Example SIS project illustrating each phase of the safety life cycle with application program development using relay ladder language) shows the usage of CEM to describe the Application Program requirements. Annex A (Guidance for IEC 61511-1) mentions the usage logic diagrams for detailed functional safety requirements for each Safety Instrumented Function (SIF). Annex B (Example of SIS logic solver application program development using function block diagrams) gives an example of how to use state machines.

Verification

PLCverif is used to verify that the specification is aligned with the code. Different discrepancies can be found during the collaboration, which can classified as follows:

- *Specification* findings. There is a disparity between the specification and the implementation. However, the implementation aligns with the intended goals of the safety engineers. The resolution entails modifying the specification.
- *Implementation* finding. The specification is accurate, but a bug exists in the implementation. The predominant issue often comes from alterations in priorities. The solution involves amending the code.
- *Documentation* finding. The meaning of the specification is correct, yet minor errors are present, like typographical mistakes (incorrect variable names) or incomplete coverage of possible scenarios. The remedy involves revising and updating the specification.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP001

GSI CASE STUDY

FAIR is an international accelerator facility dedicated to research involving antiprotons and ions. This collaborative endeavor involves international partners and is presently in development at the *GSI Helmholtzzentrum für Schwerionenforschung*.

The Personnel Access System (PAS) at FAIR [15] prevents personnel from entering areas exposed to particle beams and their radiation, so-called NE-Areas. It handles the challenge of numerous accessible zones efficiently by allowing beam operations in some areas while personnel accesses others. Additionally, for hazards like electricity, RF, and lasers, the PAS permits access only to authorized personnel and activates safety measures.

Only a few components along the beamline can quickly and specifically stop beam operation in a designated NE-Area without affecting the entire facility. These critical components are known as Important Safety Elements (ISE). A collection of ISE that halt beam operation in a specific NE-Area is called a Beam Off Group (BOG) for that area.

The PLC program designed for PAS is developed using TIA Portal [16], Siemens' proprietary platform for PLC program development. This PLC program is modular, and highly configurable, making it possible to use it in all NE-Areas by only changing its configuration variables. Model checking can verify the properties for all the configurations.

Nevertheless, since TIA Portal is a private platform, the precise workings of numerous functions are safeguarded as proprietary know-how (built-in functions). To verify the PLC program with PLCverif, it becomes necessary to conceptualize these behaviors in a structured manner. Consequently, this modeling process is intricate and labor-intensive. Particularly challenging are functions that involve timing aspects, as they require the propagation of signal values across successive PLC execution cycles.

To tackle this challenge, a method involving TIA Portal simulations was employed to understand the behaviors of these functions. Following this, the PLCverif models were subjected to verification. Using this approach ensures that the formal models conform to the behavior exhibited by the original Siemens' built-in functions, thus providing a formal assurance of their equivalence. Some of the functions whose behavior was replicated include CTUD (up and down counter), ESTOP1 (Emergency STOP/OFF up to stop category 1), F-I/O passivation and reintegration, FDB_TIME (feedback time), and FEEDBACK [17].

The collaboration between CERN and GSI identified a number of disparities between the specification and the PLC code. For the BOG PLC program, there were 9 specification findings, 8 implementation findings, and 9 documentation findings.

CONCLUSION

The collaboration model presented in this paper to apply formal verification techniques to critical PLC programs outlines a win-win situation for all the counterparts. It helps specification engineers and PLC developers deepen their understanding of their PLC programs' behaviors. It enables them to refine specifications, address errors, and bring comprehensive system enhancements. They also acquire technical knowledge from the formal verification engineers, and by the end of the collaboration, they are able to perform the verification tasks independently. Furthermore, formal verification engineers also take advantage of this collaboration through enhancements to the PLCverif tool.

The case study shown in this paper was divided between CERN and GSI according to the expertise and needs of each organization. However, it is important to emphasize that collaborations between different institutions might take diverse forms based on their strengths and objectives. The key takeaway is that engaging in partnerships with other entities to enhance the safety of systems through formal verification is both feasible and advantageous.

Such collaborations can lead to the pooling of knowledge, resources, and insights from various sources, fostering a richer understanding of safety-critical systems. By working collectively, institutions can tackle complex challenges and ensure that formal verification practices are effectively integrated into their processes. This not only enhances the safety of the systems but also promotes the sharing of best practices and lessons learned, benefiting the entire collaborative ecosystem. The potential outcomes extend beyond immediate projects, influencing the broader domain by establishing a culture of rigorous verification and safety enhancement.

REFERENCES

[1] J.-C. Tournier, B. F. Adiego, and I. Lopez-Miguel, "PLCverif: Status of a Formal Verification Tool for Programmable Logic Controller", in *Proc. ICALEPCS'21*, Shanghai, China, 2022, paper MOPV042, pp. 248–252. doi:10.18429/JACoW-ICALEPCS2021-MOPV042

[2] E. B. Viñuela, D. Darvas, and V. Molnár, "PLCverif Re-engineered: An Open Platform for the Formal Analysis of PLC Programs", in *Proc. ICALEPCS'19*, New York, NY, USA, 2020, pp. 21–27. doi:10.18429/JACoW-ICALEPCS2019-MOBPP01

[3] I. D. Lopez-Miguel, B. F. Adiego, J.-C. Tournier, E. B. Viñuela, and J. A. Rodriguez-Aguilar, "Simplification of Numeric Variables for PLC Model Checking", in *2021 19th ACM-IEEE Int. Conf. Formal Methods Models Syst. Design (MEMOCODE)*, 2021, pp. 10–20. doi:10.1145/3487212.3487334

[4] D. Giannakopoulou, T. Pressburger, A. Mavridou, J. Rhein, J. Schumann, and N. Shi, "Formal requirements elicitation with FRET", in *REFSQ Tools*, 2020. https://ntrs.nasa.gov/citations/20200001989

[5] Z. Ádám *et al.*, "From Natural Language Requirements To the Verification Of Programmable Logic Controllers: Integrating FRET Into PLCverif", in *Proc. NASA Formal Methods: 15th Int. Symp.*, Houston, TX, USA, 2023, pp. 353–360. doi:10.1007/978-3-031-33170-1_21

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP001

[6] D. Beyer, T. A. Henzinger, R. Jhala, and R. Majumdar, "The software model checker Blast", *International Journal on Software Tools for Technology Transfer*, vol. 9, no. 5-6, pp. 505–525, 2007. doi:10.1007/s10009-007-0044-z

[7] B. Fernández Adiego *et al.*, "Applying model checking to industrial-sized PLC programs", *IEEE Trans. Ind. Inf.*, vol. 11, pp. 1400–1410, 2015. doi:10.1109/TII.2015.2489184

[8] B. F. Adiego *et al.*, "Applying Model Checking to Critical PLC Applications: An ITER Case Study", in *Proc. ICALEPCS'17*, Barcelona, Spain, 2017, pp. 1792–1796. doi:10.18429/JACoW-ICALEPCS2017-THPHA161

[9] B. F. Adiego, E. B. Viñuela, F. Havart, T. Ladzinski, I. Lopez-Miguel, and J.-C. Tournier, "Applying Model Checking to Highly-Configurable Safety Critical Software: The SPS-PPS PLC Program", in *Proc. ICALEPCS'21*, Shanghai, China, 2022, paper WEPV042, pp. 759–763. doi:10.18429/JACoW-ICALEPCS2021-WEPV042

[10] Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 3: Software requirements, https://webstore.iec.ch/publication/5517

[11] B. F. Adiego *et al.*, "Cause-and-Effect Matrix Specifications for Safety Critical Systems at CERN", in *Proc. ICALEPCS'19*, New York, NY, USA, 2020, pp. 285–290. doi:10.18429/JACoW-ICALEPCS2019-MOPHA041

[12] Automated Verification of Programmable Logic Controller Programs against Structured Natural Language Requirements, https://ntrs.nasa.gov/citations/20230003752

[13] D. Darvas, E. B. Vinuela, and I. Majzik, "A Formal Specification Method for PLC-based Applications", in *Proc. ICALEPCS'15*, Melbourne, Australia, 2015, pp. 907–910. doi:10.18429/JACoW-ICALEPCS2015-WEPGF091

[14] Functional safety - Safety instrumented systems for the process industry sector - Part 2: Guidelines for the application of IEC 61511-1:2016, https://webstore.iec.ch/publication/25521

[15] D. Gaßmann *et al.*, "The personnel access system for fair", in *Proc. IPAC'23*, Venice, Italy, 2023, pp. 4067–4069. doi:10.18429/JACoW-IPAC2023-THPA049

[16] Totally Integrated Automation Portal, https://www.siemens.com/global/en/products/automation/industry-software/automation-software/tia-portal.html

[17] STEP 7 and WinCC Engineering V18 - System manual, https://support.industry.siemens.com/dl/files/056/109815056/att_1121875/v5/STEP_7_WinCC_V18_enUS_en-US.pdf