Design of Modular and Distributable Automation Software for PLCs

Oscar Miguel-Escrig[a], Isabel Roselló-González and Julio-Ariel Romero-Pérez[b]
Departament d'Enginyeria de Sistemes Industrials i Disseny, Universitat Jaume I,
av. de Vicent Sos Baynat, Castelló de la Plana, Spain

Keywords: Distributed Control System, Grafcet, IEC 60848, IEC 61131, PLCs.

Abstract: Design and maintainability of modular automation software are common concerns nowadays. Common practice in industry usually overlooks the design phase of software, jumping directly into the coding phase, which typically results in poorly readable and maintainable code. In this work, it is shown how modular and hierarchically structured design of discrete event control systems, which is supported by Grafcet models, can be subsequently implemented in several devices distributed across the fieldbus. Besides, the resulting software is more readable and maintainable due to its similarities with the proposed Grafcet model. An example is provided showing how a distributed application can be tested in a centralized fashion.

1 INTRODUCTION

Automation software in industry is mainly implemented in devices called Programmable Logic Controllers (PLC). Their usage is extended across industry due to their degree of standardization, their acceptance and accumulated know-how among practitioners and the support provided from institutions and related companies.

Different aspects surrounding PLCs are standardized in the norm IEC 61131 (IEC, 2013), which is widely adopted among the control community. PLCs from different suppliers are programmed using Integrated Development Environments (IDEs) based on the standard IEC 61131, which interpret and incorporate in their programming environment the principles of the standard, enabling portability to different extents.

It is worth mentioning that, in the context of distributed control systems, standard IEC 61499 (Zoitl and Lewis, 2014) has been proposed to implement automation software. Despite the fact that its software components are based on IEC 61131's, IEC 61499 is not as spread across industry as IEC 61131. However, remarkable efforts are being made from the industry domain and the academia to develop and integrate automation software according to IEC 61499, (Lyu and Brennan, 2021).

The standard IEC 61131 defines in its third part (IEC 61131-3) the programming languages that can be used to implement the automation software. Namely, these languages are Sequential Function Chart (SFC), Function Block Diagram (FBD), Ladder Diagram (LD), Structured Text (ST) and Instruction List (IL). The first three languages are graphical, and the remaining two textual. A project according to IEC 61131 can be composed of several Program Organization Units (POU) each of them programmed in the language of preference. Each of the languages defined in the standard has its own strengths, which can be exploited in the coding process. Despite the flexibility and variety of languages, LD and ST are the most used nowadays, (PLCopen, 2019).

Before starting coding, it is advisable to properly define the system's functionality with an appropriate model. This step is generally overlooked in industry because it is perceived as an unnecessary step that delays code development. Some studies like (Johnson, 2007; Ljungkrantz et al., 2011) point out that this underuse of models is due to strong time constraints and to a lack of tools for automatically obtain control software from models. However, coding directly from the textual definition of the system's behavior leads to possible delays and results in harder to maintain code since no formal definition of the behavior using models has been made.

There exist several tools for modeling the behavior of automation systems which have been proved in practice. Some examples of these tools are Control Interpreted Petri Nets, Grafcet diagrams or UML

[a] https://orcid.org/0000-0002-2472-2038
[b] https://orcid.org/0000-0003-3397-2239

Miguel-Escrig, O., Roselló-González, I. and Romero-Pérez, J.
Design of Modular and Distributable Automation Software for PLCs.
DOI: 10.5220/0012183700003543
In *Proceedings of the 20th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2023) - Volume 2*, pages 135-141
ISBN: 978-989-758-670-5; ISSN: 2184-2809

state diagrams. The obtained models allow a formal description of the behavior, thus avoiding the ambiguities that a textual description can induce. In this work, Grafcet diagrams, defined in the IEC 60848 (IEC, 2002), which are state-transition diagrams well-known in the automation field, will be used to model the behavior of discrete event control systems.

Grafcet diagrams and SFC language of standard IEC 61131 are similar since both have the step-transition concepts at its base. This makes the implementation of discrete event controller in SFC easier to read and understand compared with implementations of controllers in other languages of the standard. Thus, the usage of SFC results in a more maintainable code. Furthermore, due to the similarities between Gafcets and SFC, the formal description of the controller behavior in Grafcet models can be easily obtained from the SFC code, even if it is modified.

The aim of this work is to show that modular design enables for a modular implementation, enhancing the capability of the software to be distributed across fieldbus devices, maintaining the structural hierarchy of the Grafcet model. The translation step from the model to the implementation is out of the scope of the work since it has been detailed in the literature (Julius et al., 2017; Schumacher and Fay, 2014). However, the usage of SFC for implementing structuring mechanisms will be also explored. Besides, aspects like readability and maintainability will be shown to be enhanced through SFC implementation from GRAFCET diagram.

The structure of the work is as follows. In Section 2 some preliminary discussion is made with regard to the modeling and implementation of distributed automation software. In Section 3 the procedure for the design and implementation of the software is explained. Section 4 presents the platform used for validating the results, together with the modeling and implementation details. Finally, in Section 5 de conclusions about this work are drawn.

2 PRELIMINARY NOTES

2.1 Modeling

Among the different modeling techniques available, Grafcet diagrams, defined in the standard IEC 60848, are a well-known tool in the field of control engineering since this kind of diagrams form part of the contents studied in higher education and technical schools of the automation branch.

This modeling language allows the designer to define the general behavior of an automation system regardless of the particular implementation details, leaving their definition for subsequent and more detailed design rounds or directly for the implementation.

Grafcet diagrams allow defining the detailed behavior of discrete event systems through a modular and hierarchical design. This is accomplished, in part, because of the language elements, which cover most of the possible situations that can be found in actual applications. Besides, the user has the possibility of modeling the functioning of a whole system through different partial Grafcet diagrams, enabling a modular design.

To coordinate the different partial Grafcets that can result from the design process, the norm provides some mechanisms for hierarchically structuring the functioning. In particular, the modeling mechanisms are enclosures, macro-steps and forcing orders. The particularities and rules concerning these mechanisms are defined in the norm, and will not be further discussed. In this work, forcing orders will be used as structuring mechanism since they enable for a hierarchical design.

It is worth noticing that these principles of modular design, hierarchy and abstraction in the behavior definition, are not tied to a given physical device. In this sense, there is not any hindrance to use Grafcet diagrams for modeling distributed automation applications.

2.2 Implementation

The standard IEC 61131 in its third part offers five programming language for implementing automation software, each of them with their own properties which make them more suitable than another language for a given task.

In (Bonfatti et al., 1997), a discussion about the roles of the language in IEC 61131 was provided. SFC was highlighted as the most suitable to support the initial phases of PLC software development. Several characteristics that endorsed SFC were mentioned: 1) High expressive power, 2) Graphic formalism, 3) Support to preliminary design, 4) Support to detailed design, 5) Natural connection with other languages and 6) Support to software fragmentation.

Due to these characteristics, SFC is recommended for early design and to structure any application, leaving the remaining languages to auxiliary tasks as further specifying or implementing specific parts of code. According to this, a hierarchy was defined in (Bonfatti et al., 1997) between the IEC 61131 languages, regarding to their capabilities in different coding phases and to their level, which is presented in

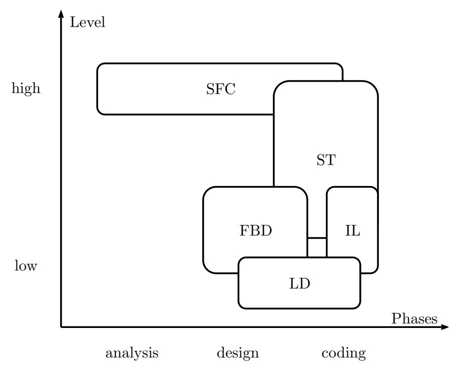

Figure 1: Hierarchy of IEC 61131 languages according to (Bonfatti et al., 1997).

Figure 1. This recommendation for the implementation will be kept in this work since it allows maintaining the structure and hierarchy defined in Grafcet models.

Since SFC language is very close to Grafcet modeling, being both of them state diagrams, the comments made regarding the applicability of Grafcet models to design of distributed automation software, also apply for SFC's implementation. However, it is important to notice that, while Grafcet models can theoretically split steps across devices, (Miguel-Escrig et al., 2020; Wiesmayr et al., 2021), SFC's implementation is tied to a single device. This aspect will have to be considered in the modeling process.

Regarding the distribution of programs developed under the IEC 61131 standard, it should be noted that, unlike the IEC 61499, the IEC 61131 is highly tied to the device where the code is implemented. This standard, however, supports communication between devices. The information exchange is supported in the standard by access paths mechanisms. Alternatively, other communication approaches can be used, like Modbus or OPC.

3 FOLLOWED PROCEDURE

The procedure followed for the development of distributable automation software consists of a design phase using Grafcet and an implementation phase using SFC as the main language, using the rest of languages of the IEC 61131 for further code developing or adding some specific feature.

In the design phase, the process starts by identifying the independent "tasks" that compose the whole system behavior. The tasks are delimited according to three principles, namely, interaction between tasks, virtual modularity and physical modularity.

Reducing the interaction between tasks allows lowering the communication network traffic, therefore, problems associated with the communication such as packet losses or delays have a lesser impact on the systems' performance.

A differentiation has been made with regard to the principle of modularity, which has been split between virtual and physical. Both aspects are equally important and they take into account different characteristics of the system. Under the physical modularity, there are gathered aspects as plant layout or re-usability of components (e.g. conveyors) in other processes. Whereas by virtual modularity, all the aspects regarding the software design and implementation are gathered such as PLCs used, programs re-usability, fieldbus constraints, etc.

With the different tasks identified following as much as possible these principles, a Grafcet model is designed for each of them. If needed, several partial Grafcets are designed to correctly characterize the behavior of the task. Afterward, a main Grafcet model functioning as a coordinator between the previous partial Grafcets is designed. This main Grafcet is in a hierarchically higher position with regard to the partial Grafcets modeling the tasks and contains structuring mechanisms like forcing orders or enclosures. In this main Grafcet are included the coordination tasks for the normal operation of the process and also stoppage or maintenance procedures.

Once the behavior of the system has been modeled, with the desired degree of precision, through different Grafcets, the implementation in IEC 61131 languages can begin. The language of choice for this work is SFC due to its similarities with the developed Grafcet models. The methodology for implementing most of Grafcet model elements in SFC has been presented in (Schumacher and Fay, 2014). Following the procedure explained in that paper the functioning of each partial Grafcet is implemented in a different POU, being the forcing orders the remaining aspect to implement since an implementation guideline has not been provided for the structuring mechanism.

For the scope of this work, it has only been considered forcing orders that empty the marking of a Grafcet and forcing orders that initialize the functioning of a partial Grafcet to a given step.

According to the IEC 61131 standard, each SFC network must always have at least one step active. Therefore, to implement the aforementioned forcing orders in a SFC that implements a forced Grafcet, we propose to include an additional step symbolizing the empty marking state. This new step, not included in the developed partial Grafcet, will be implemented as the initial step for the SFC network and will be placed before the expected initial step for the functioning.

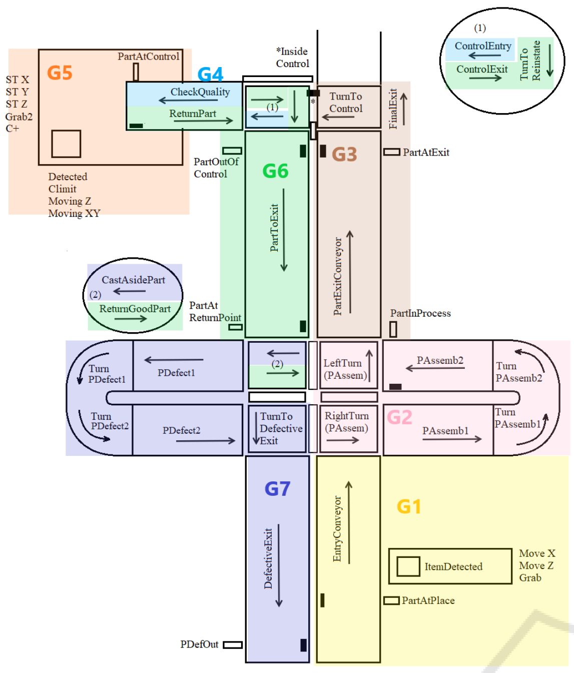

Figure 2: Diagram of the system with modeled Grafcets.

An additional always true transition will be included after the new step. Besides, some mechanism to initialize and pause the execution of SFC networks must be used. In Codesys, for instance, the flags `SFCInit` and `SFCPause` play this role (Codesys, 2022).

To implement the forcing order in the SFC that implements the functioning of a Grafcet that forces another Grafcet, it is only required to modify the state of the flags of the forced SFC accordingly. If the SFC must return to its initial step, `SFCInit` is activated and it will transition to the initial functioning step through the always true transition included. If the SFC must simulate an empty behavior, `SFCInit` and `SFCPause` must be activated.

4 EXPERIMENTAL VALIDATION

4.1 Platform Presentation

The system under study is formed by a combination of different conveyors, robots and sensors. The aim of this system is to hold an assembling process in which a lid is put onto a base. In addition, a quality control is stochastically realized upon some of the assembled pieces. The depiction of the system is presented in Figure 2.

The normal operation of the system starts when the Start Button is activated. Then, lids and bases arrive alternatively through the entry conveyor. The robot in charge of the assembly takes one lid and puts it on top of the base that comes after. Once the assembly process is done, the part goes through a group of perpendicular and round conveyors. Here, a series of manual operations are conducted on the part to obtain the final product. Then the final product is taken out of the system by another set of conveyors.

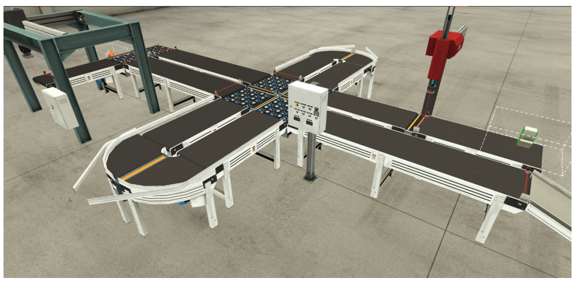

Figure 3: Digital model of the physical system in Factory IO.

The final products selected to pass the quality control are deviated towards the robot where it is carried out. This is done once the manual operation has been realized and before the piece leaves the system. Once the product has passed the quality control an operator establishes whether it has to be cast aside or not. If the product has good quality, it is returned to the normal exit. If the product has to be cast aside, it goes out of the system though another set of conveyors, where an operator examines the part to make most of the material if it can be used again. Then the part leaves the system to the defective part storage.

To stop the system, the Stop Button has to be pressed. Once this button is activated, no more parts arrive and the parts that are already into the system are processed. When all the pieces have abandoned the system, the conveyors stop and the system waits to be activated again. There is also an Emergency Button that immediately stops all the elements in the system when pressed. When the emergency is repaired and an operator releases the Emergency Button, all the processes in the system restart.

A digital model of the physical system is built using Factory IO which is shown in Figure 3. This environment allows testing the final automation software in a realistic manner, since the communication with the PLC code is performed via standardized protocols like Modubs or OPC. Besides, Factory IO already implements the physics of the components (like conveyors or robot arms), which provides a realistic feedback of the behavior. For more details the reader is referred to their website, (Factory IO, 2022).

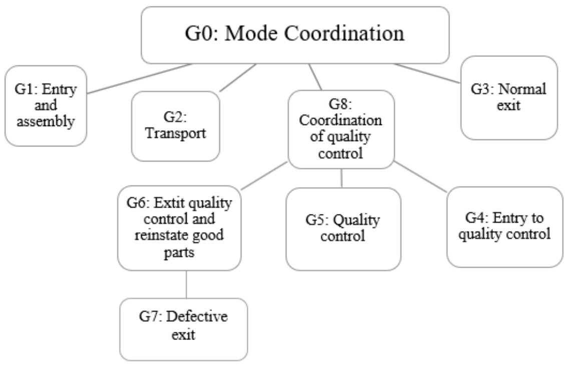
G0: Mode Coordination
G1: Entry and assembly
G2: Transport
G8: Coordination of quality control
G3: Normal exit
G6: Exit quality control and reinstate good parts
G5: Quality control
G4: Entry to quality control
G7: Defective exit

Figure 4: Identified tasks and their hierarchical relationship.

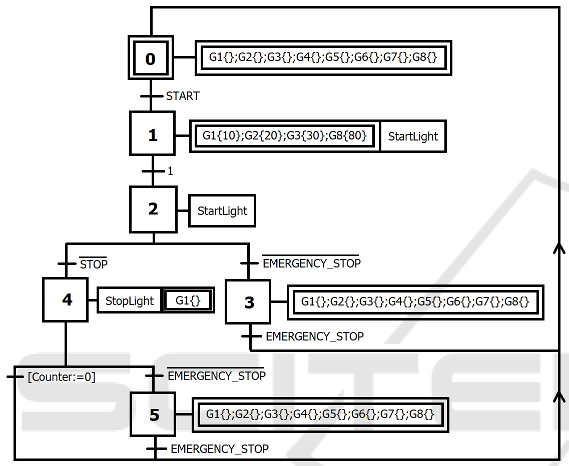
0
G1{};G2{};G3{};G4{};G5{};G6{};G7{};G8{}
START
1
G1{10};G2{20};G3{30};G8{80}
StartLight
1
2
StartLight
STOP
EMERGENCY_STOP
4
StopLight
G1{}
3
G1{};G2{};G3{};G4{};G5{};G6{};G7{};G8{}
EMERGENCY_STOP
[Counter:=0]
EMERGENCY_STOP
5
G1{};G2{};G3{};G4{};G5{};G6{};G7{};G8{}
EMERGENCY_STOP

Figure 5: Mode Coordination Grafcet G0.

4.2 Modeling Details

With the goal of reducing the communication traffic and producing modular software components, several tasks have been identified. It has been attempted that each task corresponds to a given physical module, however, with the aim of reducing the communication traffic some parts have been mixed into a single task. The resulting tasks and their hierarchical relationship are presented in Figure 4. The modularity of the tasks can be appreciated in Figure 2, where the distribution of the developed Grafcets in the physical layout is presented.

In the communications aspect, the data shared between Grafcets are the forcing orders and some flags indicating the usage of actuators in the points where bidirectional movement can happen to avoid activating both actuators at the same time.

With regard to the structure, a general Mode Coordination Grafcet (G0) has been included in the highest hierarchical position, which is presented in Figure 5. The remaining tasks are placed on the same hierarchy level, but Grafcet G8 has been developed to facilitate

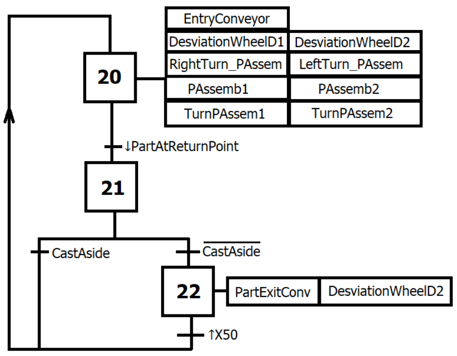

Figure 6: Partial Grafcet of task G2: Transport.

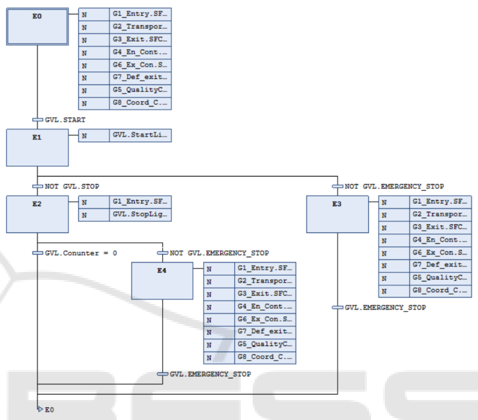

Figure 7: SFC implementation of Mode Coordination Grafcet G0 in Figure 5.

the coordination between the different tasks related with the quality control. As an example, the partial Grafcet G2:Transport is included in Figure 6.

4.3 Implementation Details

The implementation of the previously presented Grafcets has been done in separate POUs in Codesys. Concretely, these POUs are of type `PROGRAM`, in this way, the code developer can chose to distribute them across different devices.

Since modular software has been developed, there is a great flexibility in how to distribute the application. The level of flexibility is such that each POU could be executed by a different PLC, although other more reasonable and practical choices are possible.

Each of these POUs has been programmed using SFC language following the procedure explained in the previous section. Examples of the implementation are provided in Figures 7 and 8 for Grafcets G0 and G2, respectively.

It is worth noticing the remarkable similarities between the SFC implementations and the Grafcet models. This feature enhances the readability and main-

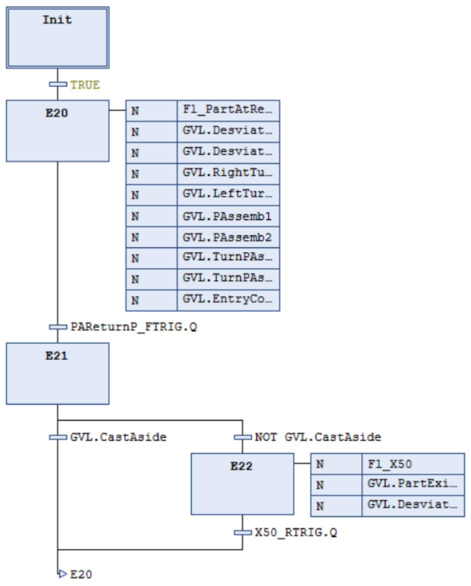

Figure 8: SFC implementation of task G2: Transport from Grafcet in Figure 6.

tainability of the developed software because, if differences in the code structure are spotted, they can be easily translated to the model documentation and vice-versa.

The communication has been established between the different POUs through an OPC UA server, where the variables referring to the state of actuators and the forcing orders are shared.

4.4 Validation

The functioning of the application can be tested in a centralized fashion, execution all the developed POUs in the same PLC. It has to be taken into account that each POU will have Local and Shared variables, which will be used by its SFC. To perform the testing, Shared variables must be declared in the OPC Server, which will provide the communication link between POUs. Feedback from the state of the system is obtained from the Factory IO model, which is also connected through OPC UA.

The described structure for testing distributed applications with Codesys and Factory IO via OPC UA is presented in Figure 9.

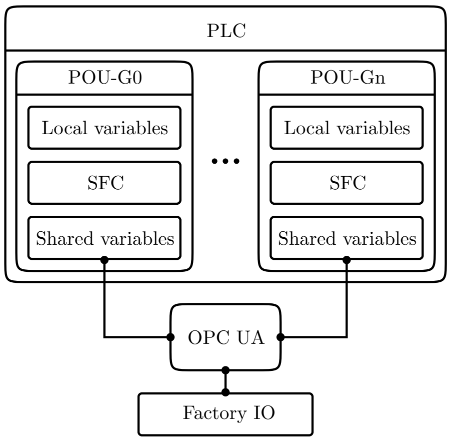

Figure 9: Communication structure for testing distributed applications.

5 CONCLUSIONS

In this work the importance of a proper design phase for modular automation software is highlighted. This step, generally overlooked in industry practice, if done properly facilitates the implementation of modular and distributable automation software.

To model the behavior of an application with these characteristics of modularity and distributability, it is proposed to use Grafcet models, which allow the design of software components with different degrees of abstraction.

The functionality that these Grafcet models describe is implemented in PLC controllers, which constitute the hegemonic device in industry and are typically programmed according to the standard IEC 61131. It has been chosen to use mainly SFC language due to its similarities with Grafcet models, resulting in an easy to read and maintain code.

Grafcet standard provides a series of mechanisms for hierarchically structuring applications. Guidelines to implement part of these mechanisms in IEC 61131 are provided.

Finally, an example is provided in which the described design and implementation procedure is applied. In that example, it is shown how the testing of the resulting distributed application can be done in a centralized fashion.

ACKNOWLEDGEMENTS

This work is supported by the project number UJI-B2021-45/21I596 from Universitat Jaume I.

REFERENCES

Bonfatti, F., Monari, P. D., and Sampieri, U. (1997). *IEC 61131-3 programming methodology: software engineering methods for industrial automated systems.* ICS Triplex.

Codesys (2022). Codesys online help - SFC flags. https://content.helpme-codesys.com/en/CODESYS Accessed: 2023-06-10.

Factory IO (2022). Factory IO - Manual. https://docs.factoryio.com/manual/. Accessed: 2023-06-10.

IEC (2002). IEC 60848: GRAFCET specification language for sequential function charts.

IEC (2013). IEC 61131 - programmable controllers, part 3: Programming languages.

Johnson, T. L. (2007). Improving automation software dependability: A role for formal methods? *Control engineering practice*, 15(11):1403–1415.

Julius, R., Schürenberg, M., Schumacher, F., and Fay, A. (2017). Transformation of grafcet to plc code including hierarchical structures. *Control Engineering Practice*, 64:173–194.

Ljungkrantz, O., Akesson, K., Yuan, C., and Fabian, M. (2011). Towards industrial formal specification of programmable safety systems. *IEEE transactions on control systems technology*, 20(6):1567–1574.

Lyu, G. and Brennan, R. W. (2021). Towards iec 61499-based distributed intelligent automation: A literature review. *IEEE Transactions on Industrial Informatics*, 17(4):2295–2306.

Miguel-Escrig, O., Romero-Pérez, J.-A., Wiesmayr, B., and Zoitl, A. (2020). Distributed implementation of grafcets through iec 61499. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 402–409.

PLCopen (2019). PLC Programming Preference Survey - Insights and User Comments. https://www.automation.com/en-us/articles/2019/plc-programming-preference-survey-insights-user-co. Accessed: 2023-06-10.

Schumacher, F. and Fay, A. (2014). Formal representation of grafcet to automatically generate control code. *Control Engineering Practice*, 33:84–93.

Wiesmayr, B., Zoitl, A., Miguel-Escrig, O., and Romero-Pérez, J.-A. (2021). Distributed implementation of hierarchical grafcets through iec 61499. In *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8.

Zoitl, A. and Lewis, R. W. (2014). *Modelling control systems using IEC 61499*, volume 95 of *IET Control engineering series*. IET, London, 2. ed. edition.