

Article

Clarifying and Extending IEC 60848 for Unambiguous GRAFCET Implementation in Embedded Systems

Angel Gaspar Gonzalez-Rodriguez , Elisabet Estevez-Estevez , Jose Vicente Muñoz-Diez ,
Pedro Jose Casanova-Pelaez  and Joaquin Cañada-Bago 

Department of Electronic Engineering and Automation, University of Jaen, 23071 Jaen, Spain;

eestevez@ujaen.es (E.E.-E.); jmunoz@ujaen.es (J.V.M.-D.); casanova@ujaen.es (P.J.C.-P.); jcbago@ujaen.es (J.C.-B.)

* Correspondence: agaspar@ujaen.es

Abstract

This work proposes guidelines for coding a grafcet on a microcontroller, following the specifications described in standard IEC 60848:2013. While previous works exist for coding a simple grafcet on platforms like Arduino or STM32, they overlook implementing timers, hierarchical structure, and other advanced resources included in the IEC 60848:2013. Throughout this study, procedures for implementing the majority of the resources described in the standard using pseudocode and C++ are presented. These resources include timers, stored actions, macrosteps, enclosures, and forcing orders, among others. Previously, it has been necessary to identify ambiguities, vagueness, inaccuracies, and the non-modular approach of the IEC 60848:2013 standard that impeded a proper and unequivocal implementation, proposing improvements and interpretations to make it more unequivocal and exportable. Examples are included to illustrate how to apply the proposed guidelines, and a template to assist in coding any grafcet is made available in a repository. As a result, this article provides designers with a complete set of tools and resources to unequivocally implement concurrent discrete-event processes. The implementation on several microcontrollers exhibits exceptional memory efficiency under scaled stress-testing conditions (100 steps and 25 timers): less than 2% of the RAM for STM32F411RE and ESP32, and less than a 1.5% increase in ROM when implementing the analysis of 100 transitions and the definition of 25 timers. This proves its scalability for complex discrete-event applications. Cycle times depend on whether there is access to macrostep expansions and enclosures but are around 80 microseconds for the STM32F411 and 20 microseconds for the ESP32.

Keywords: GRAFCET; embedded system; IEC 60848; metamodel; enclosing step; macrostep; forcing orders; stored actions



Academic Editor: Diego
Langarica-Córdoba

Received: 24 May 2026

Revised: 23 June 2026

Accepted: 25 June 2026

Published: 30 June 2026

Copyright: © 2026 by the authors.
Licensee MDPI, Basel, Switzerland.
This article is an open access article
distributed under the terms and
conditions of the [Creative Commons
Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

1.1. Motivation

With the growing operational capacity of microcontrollers as part of embedded systems, the number of tasks executed simultaneously is also increasing. An example may be the case of an embedded system with a user interface, which must not lose control of the system while the interaction with the user is attended to. There are works that address the problem of attending to a graphical interface while controlling discrete events, such as in [1,2], which use reactive synthesis as a technique to define the behavior of a controller in response to system inputs or operator actions. In [3], the authors model the problem using process algebra and bisimulation-based semantics. As a drawback of both solutions, they

require a formal specification of the controller's behavior in a language that is not familiar in the design of industrial automation systems.

The utility of GRAFCET (GRAPhe Fonctionnel de Commande Étape/Transition) as a tool for defining and resolving discrete automatisms is well-established [4], offering a simple visual description of process sequences and facilitating implementation of multiple threads of work simultaneously in a natural way. In projects involving microcontrollers that are not being programmed under a real-time operating system, the use of grafquets and, to a limited extent, interrupts represent the only way to manage multiple threads of work.

In this case, a grafquet (we will use GRAFCET for the standard and the modeling language, and grafquet for a particular solution using GRAFCET modeling language) constitutes a suitable alternative to real-time operating systems. This is complemented by being a systematic tool for describing and solving discrete automations which is very familiar to control engineers. However, the lack of support and guidelines for using GRAFCET on such platforms discourages programmers from employing it in discrete-event automation solutions, despite its outstanding suitability and advantages for these types of systems.

The IEC 60848 standard [5] represents an advancement in terms of standardization, as it allows for actions and transitions represented in GRAFCET to be unequivocally codified in most cases, enabling nearly any programming platform to implement GRAFCET. This International Standard establishes the GRAFCET specification language for describing the functional behavior of the sequential component within a control system. It outlines the symbols and regulations governing the graphical depiction of this language, along with guidelines for its interpretation.

As an essential component of large-scale project representation, it also encompasses the description of structures enabling hierarchy, structuring, and documentation of the code, such as forcing orders, enclosing steps, and macrosteps, which allow for the control of multiple processes as well as the establishment of supervisory and monitoring roles, necessary for the proper performance of graphical interfaces. The GEMMA guideline is not conceived without the use of these structures, as can be seen in [6].

However, as far as the authors know, there is currently no known method to fully and unequivocally translate all the structures and resources of GRAFCET into any specification language. Practical implementation of this standard reveals numerous ambiguities and vaguenesses, especially when dealing with hierarchical structures, transient steps, and stored actions.

1.2. State of the Art

As pointed out in [7], an ambiguous definition of the IEC 60848 specification can lead to different behaviors if different authors have interpreted the ambiguity in different ways. One source for this imprecision is when the same condition appears as the transition at consecutive steps, or after a sequence of transient steps (TSs). Provost in [8] presents a pseudocode that only allows for firing a transition at an execution cycle, which solves this ambiguity. This constraint is not usually a concern when the execution time (typically microseconds or a few milliseconds) is faster than the occurrences of events [4,9].

Before discussing the development of a method or tool to translate GRAFCET into a programming language, many authors have recognized the need to address these shortcomings of the standard. Provost in [10] proposes formal semantics for translating some non-timed GRAFCET structures, including macrosteps, which can be used for verification purposes or to build conformance test sequences.

Julius, in [11,12], outlines two approaches for handling hierarchical structures. In these works, authors propose a procedure to automatically generate code from grafquets, tackling

hierarchical structures, and continuous and stored actions, although they do not mention other resources such as timed actions.

Other works exist that generate code compatible with PLCOpen XML, as an intermediate stage prior to its translation to another language defined by IEC61131:3 or as an input file to be imported by applications such as Codesys. In this way, Castillo et al. [13] use a method quite similar to that of Barbieri in [6] to generate code compatible with PLCOpen XML, although in both cases they do not specify the basics of this translation.

Couffin in [14] presents a conceptual metamodel to represent, in a suitable formalism, the concepts and rules of GRAFCET. Based on it, Julius in [15] defines a more concrete metamodel for an unambiguous specification of a grafcet to be used in graphical representations, and for syntax verification. This work includes the description of the classes used for this metamodel, which have been later used in [16] for the transformation of GRAFCET into GAL (Guarded Action Language) for verification purposes and syntactic checking.

Building upon previous efforts to formalize and evaluate the sources of vagueness in the standard, various works have been published presenting methods for automatically generating PLC code compliant with this norm. However, most of them only consider subclasses with very basic elements of a non-hierarchical and non-time-dependent grafcet, as indicated by Schumacher in [17]. This author, however, addresses more complex structures by decomposing them into basic ones.

The standard IEC 61131:3 [18] is more specific in the implementation details of different programming languages, mainly SFC. Starting from the third edition, it includes object-oriented (OO) features such as classes and function blocks. This OO paradigm is also utilized by a number of works that provide methods for implementing grafcets following the guidelines of this standard [19,20]. The standard also improves the definition of timers, both standalone and as part of action qualifiers. Bauer et al. in [21] provide a new mathematical formulation to clarify the sources of imprecision, among others, for the SD and SL timed actions. The remaining sections mainly address the specification of syntax for different languages, semantics, variable types, editing resources, and class structures while leaving unresolved issues such as an unequivocal implementation of TSs. Table 1 summarizes the functionalities and scope of these references compared to the ones proposed in the present work.

These representation and specification efforts do not constitute a complete guide for coding the resources described in IEC 60848, particularly when the implementation is performed in a language and platform other than PLCs. Consequently, although GRAFCET is suitable for modeling discrete processes, its application is mostly confined to PLC programming used in purely industrial settings, for controlling large, expensive, and complex systems in harsh environments with demanding robustness requirements, where PLCs remain the preferred option over other types of programmable controllers.

On the other hand, in discrete-event automated systems such as home automation, security systems, energy management, consumer electronics, automotive systems, healthcare and medical devices, agricultural automation, security and surveillance systems, and service robots, microcontrollers are often the first choice due to their size, price, and ease of connection with all types of sensors. However, even for discrete processes, GRAFCET is rarely adopted in embedded systems, although exceptions exist, such as [22], where three partial grafcets are used to control a volume-cycled mechanical ventilator.

In such cases, programmers often rely on ad hoc techniques, using non-standard tools to organize tasks and interrupts to handle concurrent events, such as user interface servicing. Exceptions exist, but only for simple cases. For instance, Comlan [23] applied it with Arduino, Puxan [24] on a generic platform, and Ndenoka [25] with a dsPic. In [26], IL instructions from IEC 61131:3 were translated into assembly for MCS-51, AVR, and STM32,

although without considering GRAFCET specifications from IEC 60848. FPGA devices have also been the target of very simplified implementations, as in [27]. Overall, none of these works explore in depth the resources of IEC 60848, restricting applicability to simplified systems. Table 2 summarizes the functionalities and scope of these reference-focused microcontrollers compared to the ones proposed in the present work.

1.3. Contributions

Motivated by this gap, this work provides resources and guidelines for implementing on a microcontroller almost any discrete event problem defined using the items in IEC 60848. Furthermore, it identifies and addresses sources of vagueness in the conceptual specifications of IEC 60848, some of which are resolved through the higher level of specification in IEC 61131:3.

Once deficiencies in the definition of IEC 60848, the gap between this norm and IEC 61131:3, and the absence of works addressing GRAFCET programming for microcontrollers (except for very basic cases) have been observed, this article makes the following contributions:

- Identifying and addressing sources of vagueness in IEC 60848.
- Proposing actions to bridge the gap between this standard and IEC 61131:3, mainly regarding the definition of timed actions, but also the invoking of function blocks (FBs) including a PG, or the implementation of transient steps. In other cases, the aim is rather to explain why the IEC 61131-3 standard modifies or even suppresses some features of the IEC 60848 standard. These two items are represented in the last row of Table 1.
- Offering examples with advanced structures comprising all resources described in the norm, where the explanation of each resource is accompanied by detailed pseudocode describing its operation. The corresponding code is also made available as examples in C++ for use as templates for user-defined applications.
- Providing a tool, in the form of a reusable C++ project with a template, that allows for the implementation of any grafcet following the standard IEC 60848. At this stage, it has been tested on microcontrollers STM32, ESP32 and Arduino DUE, although the structure is prepared to accept any other platform. These two items are represented in the last row of Table 2.

It is worth noting that the purpose is not to establish a new semantic formulation of the elements that give rise to imprecision or ambiguity but rather to facilitate the implementation of GRAFCET.

The remainder of the paper is organized as follows: Section 2 presents ambiguities found in the standard and guidelines to solve them; Section 3 describes the structure of the project implementing a grafcet in an embedded system and the platform in which the project has been compiled and tested; Section 4 explains three global grafkets with hierarchical relationships and advanced resources; and, finally, in Section 5 the conclusions of the work are released.

Table 1. Comparison of the resources and functionalities covered by different references.

Ref.	Formal Semantics	Automat. Code Gen.	Timed Actions	Hierarchy	Stored Actions	Condit. Continuous Actions	Transient Evolutions	Call to FB w. Grafcet	Classes Diagram	Complex Example
[6]				Macrostep and enclosure						Yes
[11]	Yes			Yes	On activation	Yes				Yes
[12]		Yes								
[17]	Yes	PLCopen XML		Via normalization						
[13]		PLCopen XML		Yes			Yes		Yes	Yes
[14]	Yes									
[15]		Yes							Yes	
[16]	Yes			Enclosure					Yes	
[21]	Yes		Yes							
[28]				Implem. in 61499						Yes
This work			Yes	Yes	Yes	Yes	Yes	Yes	Yes	

Table 2. Comparison of different microcontroller implementations.

Ref.	Automat. Code Gen.	Lang.	Microcontr.	Timed Actions	Hierarchy	Stored Actions	Condit. Continuous Actions	Transient Evolutions	Call to FB w. Grafcet	Classes Diagram	Complex Example
[23]	Yes	C++	Arduino								
[24]		C	Generic								
[25]	Yes	C	dsPic								Yes
[26]	From IL	C	8052 ATmega16 STM32F103								
This work		C++	STM32F411 STM32F103 ESP32 Arduino Due	Yes	Yes	Yes	Yes	Yes	Yes	Yes	

2. Ambiguities and Shortcomings of IEC 60848

Although the meaning of the numerous terms related to the GRAFCET terminology used in this work is explained in [5], before analyzing the standard and describing the proposed tool, it is worth defining three concepts:

Connected grafcet chart (CGC).

The structure in which there is always a continuity of links (alternation of steps and transitions) between any two elements, step or transition, in the grafcet chart.

Partial grafcet (PG).

A set of one or more CGCs that represents a task differentiated from the others.

Global grafcet (GG).

A set of one or more PGs representing the operation of a discrete automation controlled by a single programmed control unit.

Below are a series of ambiguities, sources of vagueness, and shortcomings of the standard identified by the authors, along with a proposal for improvement to make this standard more easily transcribable.

Some of these shortcomings actually do not correspond to standard IEC 60848 but to standard IEC 61131:3, or rather to a procedure for linking these standards. This gap between both standards has been highlighted in this work, proposing actions for an adequate transcription to one of the languages covered by standard IEC 61131:3. The following deficiencies are presented, organized as follows:

- **A lack of clear specification:** in the definition of actions (Section 2.1); in task scheduling (Section 2.2); with regard to transient evolutions (Section 2.3); and in the management of concurrency (Section 2.4).
- **Bridging the gap between standards:** in the specification of timing actions (Section 2.6) and in the pursuit of modularity (Section 2.5).

2.1. Lack of Clear Specification for the Use and Implementation of Actions

Two basic tasks are involved in representing an automation problem using GRAFCET: identifying the conditions that cause it to transition from one step to another, and acting on the outputs to perform the desired actions. However, focusing on the second task, this is not covered either by the methodological and conceptual guidelines of the IEC 60848 standard, nor by the implementation specifications of the IEC 61131-3 standard. The two main modes used are described below.

2.1.1. Continuous Actions

Continuous actions are binary actions whose states are associated to the logical value of some variable. In its simplest case, it is associated to a single step, but can also be a logical combination of steps with additional conditions, or, e.g., the state of a timer.

Specifying how to implement the coding of continuous actions is not among the objectives of standard IEC 60848, and it is not straightforward to find in IEC 61131:3 either. However, IEC 60848 establishes that this type of actions is accomplished by using **assignment on state**. Although many programmers prefer another type of assignment (see Stored Actions in Section 2.1.2), the equality assignment on state is more aligned with the guidelines provided in IEC 61131:3 and allows for decoupling the coding of the grafcet evolution from the output assignments. According to this approach, each output is tracked by evaluating the conditions that make it true. When an output is activated in multiple steps, these are combined using OR logic to assign the output.

Figure 1 represents its implementation using pseudocode. As shown, it is straightforward to include assignment conditions (symbol 22 in the norm, and **Cond** in the figure).

Since the assignment is state-associated rather than event-triggered, and given that the assignment may result from the combination of multiple steps, it is unfeasible to simultaneously encode continuous actions while analyzing transitions. This separation is also illustrated on the right side of Figure 1, which decouples the grafcet evolution coding (guided by transition analysis) from the action implementation coding.

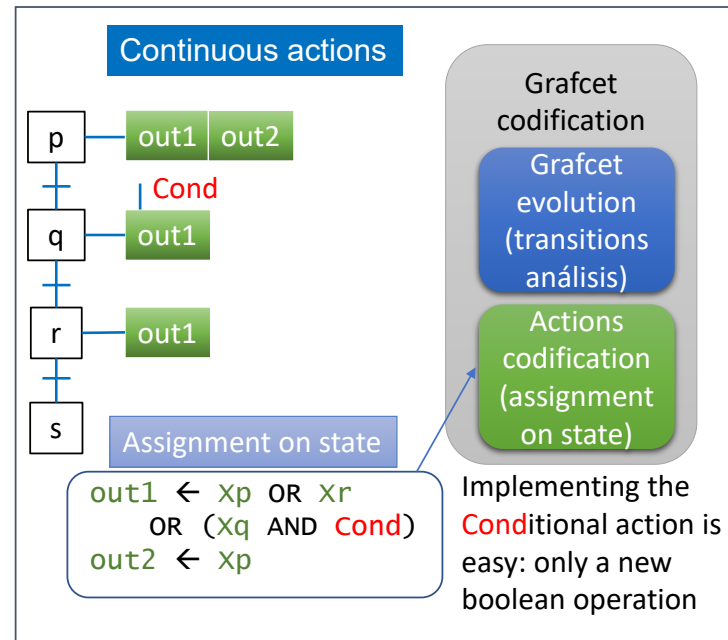


Figure 1. Encoding of continuous actions with assignment on state. There can be actions that depend on two or more steps (out1), or actions that depend on a step (or several ones) and an additional condition (Cond). Its implementation independent of the transition analysis (see boxed text at the right) simplifies coding.

2.1.2. Stored Actions

This is the second approach to setting the value of an output. In this case, the value is set by allocation, also referred to as storing, which assigns a specific value to an output or internal variable at a certain moment. The value can be a constant value or the result of a function or a logical or an arithmetical expression. The standard specifies (as allocation rule 4.8.3 in the IEC 60848) that storage occurs in response to an internal event, as opposed to continuous actions which have an assignment on state.

They are especially suitable for operations such as incrementing/decrementing variables or, in general, when assigning the result of an arithmetic operation.

Three types of stored actions are possible. They are shown respectively in Figure 2: action on step activation (symbol 27), action on step deactivation (symbol 28), or action on event (symbol 29). The first two are coded within the grafcet evolution as a response to the event that activates a step to which the action is associated.

In the third case of stored actions, the allocation takes place when, being at a certain step, an event occurs. Therefore, the assignment also depends on the step state, and it is codified separately from the transition analysis, as occurs with the continuous outputs, but using stored actions (initialization, set, reset, or increment/decrement). Ultimately, it combines both types of approaches and requires the use of edge-detection instructions to detect these events, disrupting the structure of the GRAFCET coding to provide solutions to infrequent situations, for which IEC 60848 offers alternatives. It is also worth mentioning that the symbol (29) is defined in a different way in Section 4.7.4 of this same norm.

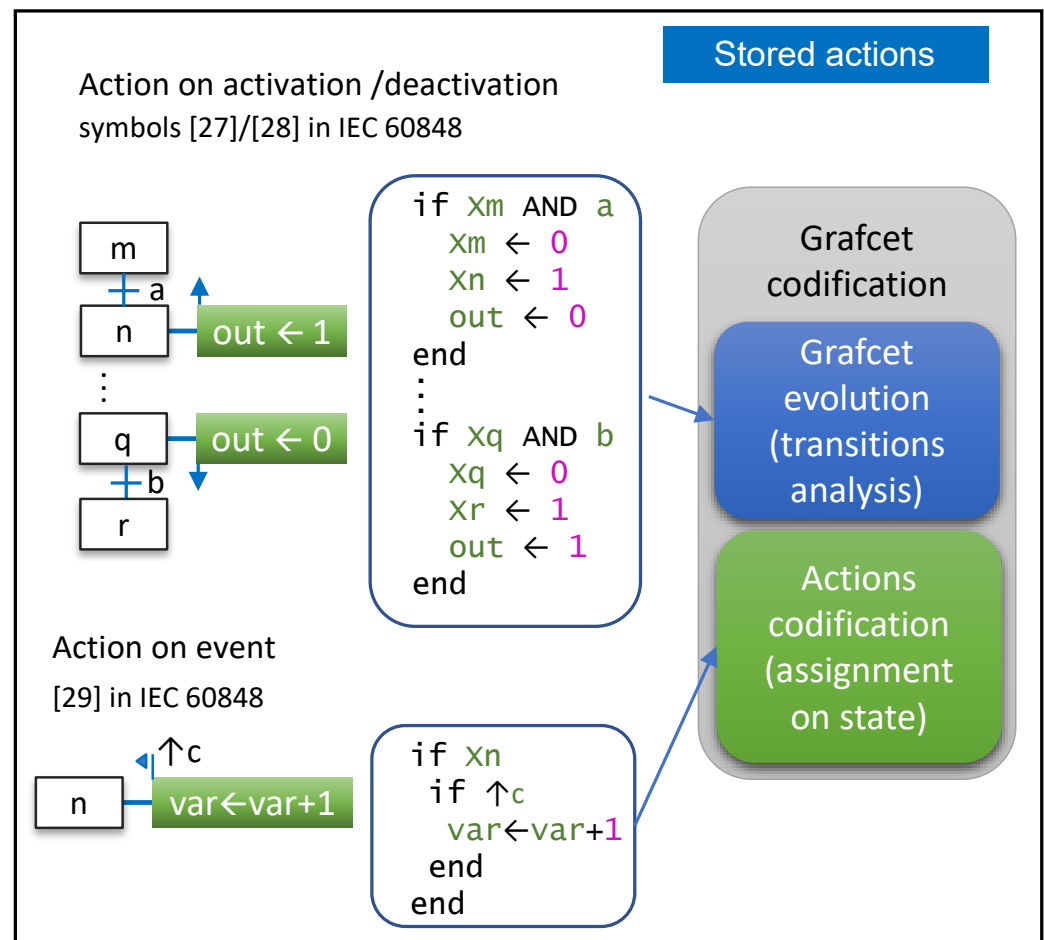


Figure 2. Types of stored actions and their implementation: action on activation, action on deactivation and action on event

2.1.3. Differences Regarding Stored Actions in Both Standards

Two important differences arise when analyzing stored actions in both standards:

- Due to the difficulties observed when dealing with event-driven actions, the IEC 61131-3 standard does not explicitly include this type.
- IEC 60848 does not consider the possibility of an output triggered by activation or deactivation that is not of the stored type but instead lasts only for a single pulse. IEC 61131-3 removes this restriction and states, for the P1 and P0 equivalents, stating that “The value of A [activation] will be TRUE for only one execution of an action called by a P1 or P0 qualifier.”. This feature is useful when marking the completion of a function block, as discussed in Section 2.5.3.
- IEC 61131-3 allows for set and reset actions while a step is active; in IEC 60848, this stored action was only defined to occur when a step is activated or deactivated (or alternatively as an action on event).

The first two items show that IEC 61131-3 provides a clearer and more complete definition of actions triggered by transition clearing, which are no longer associated with stored actions. The third one extends the possibilities for assignment of actions.

2.1.4. Continuous Actions vs. Stored Actions

As represented in Figure 2, actions on activation and actions on deactivation can be codified together with the transition analysis. With this approach, an output that must be activated from step n to step q is codified through a Set allocation when an action is executed for the first time (n) and a Reset allocation in the last step before becoming inactive

(q). Despite its seemingly higher complexity, this option is preferred by beginners since it can be coded simultaneously with the grafcet evolution (by reviewing transitions), as represented on the right of Figure 2.

In general, for non-beginner engineers, the first option (assignation on state) is preferred for several reasons: better handling in cases of convergence or divergence; direct clearing of actions in encapsulated grafcet when the encapsulating step becomes inactive; fewer code lines when programming in ST, mainly for conditioned actions; faster tracking in case of malfunction; and easier reinitialization in cases of frozen stops with output override, or, more generally, when the outputs involved in a grafcet are modified by a hierarchically superior grafcet.

The second option may be better suited for actions maintained across a long sequence of steps (e.g., operating a monostable valve for gripping or suction in a prolonged manipulation). However, including actions conditionally requires new `if . . . end` clauses (at least three for the conditioned action in Figure 1), unnecessarily complicating the code.

The following subsections highlight specific situations in which IEC 60848 does not provide clear indications for their implementation as intended by IEC 61131:3, and some guidelines are provided. The issues of task scheduling and function invocation are also addressed, since IEC 60848 defines the logical behavioral semantics of the system but does not specify its implementation or coding semantics.

2.2. Task Scheduling

Although IEC 60848 does not define a specific sequencing for transition evaluation or output assignment, its implementation can be easily adapted to a sequential execution program, even when using a single execution thread. In a programmable logic controller, the basic execution cycle (excluding interrupts and industrial communication services) consists of the following steps:

- Reading the inputs and copying them into the process image memory (PIM);
- Executing the control program, updating the system state and assigning intermediate output variables in the PIM;
- Copying the intermediate output variables from the PIM to the physical digital outputs.

This execution structure can be readily adapted to an embedded system. If communication interfaces such as SPI, I²C, WiFi, etc., are not required, the control logic can be robustly implemented in a single thread, without the need for a real-time operating system (RTOS).

Despite the sequential nature of the execution cycle, the behavior specified by the GRAFCET model remains deterministic. The evaluation of transition conditions and the update of step activation are performed during each scan cycle using the process image memory (PIM), ensuring that all transitions are evaluated based on a consistent system state. As a result, the logical concurrency defined in the GRAFCET specification is preserved at the semantic level, even though the implementation follows a sequential execution scheme.

2.3. Transient Evolution

Transient evolutions refer to the case in which *the transition conditions associated with the subsequent transitions are already true, when the first transitions considered are cleared* [5]. Typical cases of transient evolutions are the ones represented in blue in Figure 3. In this figure, the steps (Xn) are considered as unstable.

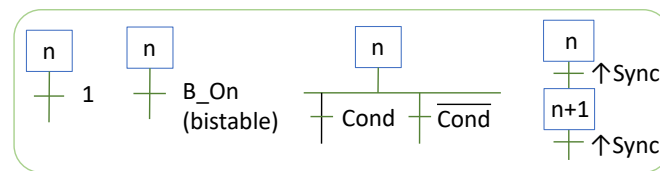


Figure 3. Different cases of transient evolutions. The second situation will be a transient one if the bistable switch is in the ON position. The fourth one is not intended as transient evolution but can behave as such depending on the implementation.

In a transient evolution, continuous actions will not be executed, or they will be executed for only one machine cycle; therefore, the actions associated with unstable steps are stored actions. IEC 61131-3 establishes that *testing of the successor transition condition(s) of an active step shall not be performed until the effects of the step activation have propagated throughout the program organization unit in which the step is declared* [18]. This holds true if the stored actions (action on activation/deactivation) are coded at the same time as the clearing of the transitions. This ensures that the assignment is performed even if the unstable step is immediately deactivated.

With regard to the fourth situation in Figure 3, which is usual in *shift registers* [5], an inappropriate transcription of a grafcet that repeats the same condition in two consecutive transitions may lead to transient evolution and unstable behavior of the step located between them. If the condition is of a level type, this situation will be unavoidable and will hinder the proper functioning of the chain. If it is an input event triggered by a edge-triggered condition, as shown in the figure and which is usual in *shift registers* [5], it is necessary to remember rule 5 of IEC 60848: *If during the operation, an active step is simultaneously activated and deactivated, it remains active* [5].

Practically, this implies analyzing the set of transitions of this grafcet connected as a sequence

```
if { } else if { } ... end_if,
avoiding the use of sequences
if { } end_if ... if { } end_if.
```

2.4. Management of Concurrency in Both Standards

Other sources of ambiguity when coding a grafcet compliant with IEC 60848 arise when dealing with exclusive (selected) branches and simultaneous branches.

In the first case, ambiguity occurs if the conditions associated with the transitions that initiate an OR divergence may become true at the same time. In this situation, IEC 60848 states that the designer should ensure that the timing, logical, or mechanical aspects of the transition conditions are mutually exclusive [5]. If this requirement is satisfied, the proper selection of conditions avoids ambiguity in the selection, and no additional clarification is required for coding. However, IEC 61131-3 specifies that the evaluation of conditions shall be performed from left to right unless an explicit priority is defined for their evaluation.

In the case of simultaneous branches, it may occur that one branch contains a stored action assigning a value of 1, while the parallel branch contains another stored action assigning a value of 0. In such a situation, the output will ultimately take the value resulting from the last action evaluated before completing the execution of the main PROGRAM POU. This issue is not fully addressed by IEC 61131-3, since it allows for the use of S and R actions, which implies a continuously stored action. The problem can be resolved if, for simultaneous branches sharing the same output, continuous assignment is used instead of stored assignment, at least for that output. This once again highlights the superior behavior of continuous assignments compared to stored ones.

2.5. Non-Modular Approach

2.5.1. Anticipating the Necessity of Modularity

The implementation as a control program of any non-trivial project across any platform (robot, microcontroller, and microprocessor) is inconceivable without the utilization of functions. In the case of PLC programming where a modular structure with a reduced number of global variables is pursued, the use of function blocks (FBs) is imperative. This is not only to provide combinational outputs based on a set of inputs but also to host secondary PGs. While the general nature of the standard does not dictate the structuring of the code implementing GRAFCET, the specifications presented, and particularly the examples found in the standard, appear to diverge the programmer from modular coding based on task division. Furthermore, there is no consideration for the possibility of executing the same PG simultaneously at two different points within the same invoking grafcet. If global variables are not utilized in the definition of PGs, there is no programming or conceptual limitation prohibiting the simultaneous execution of multiple instances of the same expansion or multiple instances of the same set of enclosures.

2.5.2. Improvement Proposal

Although the deficiencies observed in this subsection should be addressed in a more specific standard (i.e., IEC 61131:3), the following modifications are proposed to guide the programmer towards using FBs in the transcription of GRAFCET:

- The fact that the designation of steps is never repeated within a GG undesirably suggests the use of global boolean variables associated with each step, even if they belong to different grafkets. This discourages the programmer from using FBs. The examples used in the standard should be numbered as G1.0, G1.1, G1.2, ... for grafket G1; as G2.0, G2.1, G2.2, ... for grafket G2; and so on for other PGs, as done by [29]. Although the level of description pursued by IEC 60848 cannot recommend the use of FBs (as IEC 61131:3 might), this designation of steps brings forth the idea of using the same local variables, with the scope restricted to each grafket, which guides the programmer towards the use of FBs.
- There should be some indication that the same expansion can be associated with several macrosteps, allowing for the simultaneous execution of multiple instances of the same expansion. Similarly, it should be indicated that the same set of enclosures can be associated with several enclosing steps. This also leads to consideration of using FBs as a means of associating different objects with the same GRAFCET structure.

2.5.3. Call to Function Blocks

Standard 60848 does not mention the existence of functions and logically cannot provide details on their implementation. However, it includes aspects of hierarchy, from a supervisory (forcing orders), functional (enclosures), or functional/documentary (macrosteps) perspective, which are meaningless without the use of functions or, preferably, function blocks. Functions and function blocks are described by standard 61131:3, from the perspective of their structure, although it also makes no mention of the mechanism for calling and verifying their completion.

In order to utilize the powerful resource that function blocks (FBs) represent, this article proposes the calling and verification mechanism shown in Figure 4, where the state of the steps must be stored as static variables (step _X0 initially set to true), as well as the counter variables and timer blocks. It should be noted that the exit marker that attests to the completion of the process (_done) is only active during one pulse, which is not allowed by standard 60848 for action on deactivation.

2.6. Covering the Gap Between IEC 60848 and 61131-3: Managing Timings

The fundamental difference between these standards lies in their intended use: IEC 60848 is centered on the graphical description of sequential control systems using GRAFCET, primarily for specification and documentation purposes, whereas IEC 61131:3 focuses on the programming of programmable logic controllers (PLCs), offering a set of standardized programming languages, including structured text, ladder diagram, and a version of Sequential Function Charts (SFCs) adapted for implementation.

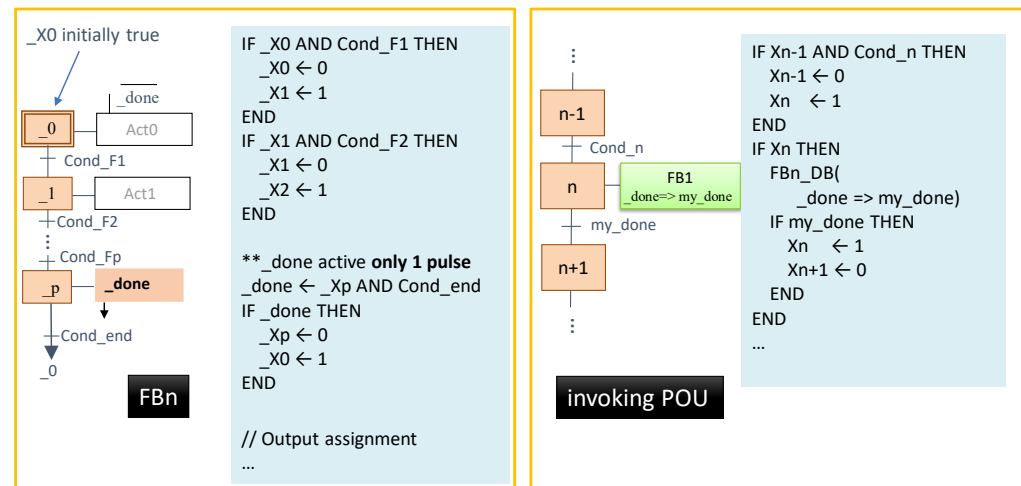


Figure 4. Call mechanism and completion acknowledgment for a function block (FB). Underscored variables are local static ones or function parameters.

Although the focus of the standard is different, IEC 61131:3 introduces some changes in the graphical representation compared to IEC 60848, although they are not always justified. The main changes appear in timed actions but also in the management of function blocks, selection of sequences, simultaneous sequences, stored actions and transient steps. These changes are detailed below, along with how to encode them using a PLC-specific language (in this case, ladder diagram to maintain a graphical approach in the explanations).

2.6.1. Global Differences Between Norms for Timed Actions

Among the resources commonly used in discrete event systems, and later only partially resolved by IEC 61131:3, timed actions entail the resource most poorly addressed in the IEC 60848 standard. It defines a generic time-dependent assignment condition in the form $t1/*/t2$, where $t1$ is the on-delay time and $t2$ is the off-delay time. Additionally, like any other action, a timed action depends on the active/inactive state of the associated step, hereinafter referred to as the Action-Execution Step (AES). The asterisk (*) shall be replaced by the timed variable, for example, an input variable or, more typically, a step variable, usually corresponding to the owning AES.

In IEC 61131:3, conditioned actions as such do not exist. Instead, a set of qualifiers is defined which can, in most cases, replace the timed actions specified in IEC 60848. The key differences are as follows:

- IEC 61131:3 does not directly allow for an external signal or a step different to the AES to condition the state of the associated output. This is also stated in IEC 60848 for delayed action and time-limited action. If the input variable were different, the number of cases to be analyzed would greatly complicate a systematic study.
- In IEC 60848, there is no direct mechanism to extend the validity of a timed action beyond the scope of the associated step. An extension of the action types defined in

IEC 60848 is studied and proposed, aiming to overcome the limitation described in the second point, specifically for timed actions controlled by the step itself.

- IEC 61131:3 is more comprehensive in defining concepts related to timing. On the one hand, it introduces six timed actions, and on the other hand, it describes the timers TON, TOF and TP, which are commonly used in modern PLC programming languages.

It is worth mentioning that neither standard explicitly describes a continuous association using negative logic.

2.6.2. Coding and Representing the Different Types of Timed Actions

One of the shortcomings found in standard IEC 60848, which is not completely addressed by IEC 61131:3, lies in the fact that time-conditioned actions are not always easily codable. It is true that most of the timing operations used are on-delay timers, for which both standards provide a simple representation and a straightforward adaptation to the desired language. However, this is not the case for other timing types.

Figure 5 lists the timed actions included in IEC 61131:3 with their timing diagrams, an implementation in LD, and their representation according to this standard and also to IEC 60848. The objective of this figure is to relate the timed actions described in IEC 61131:3 with the timers (TON, TOF, and TP) and a representation following IEC 60848.

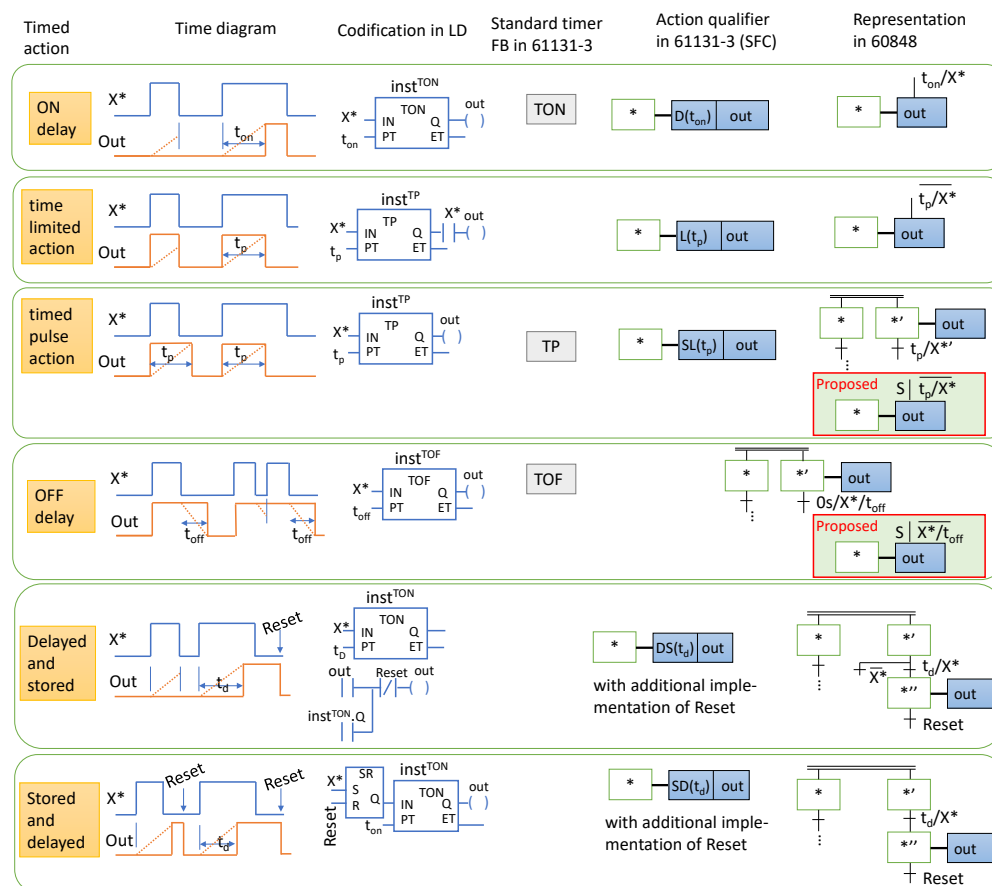


Figure 5. Timed actions in sequential control systems, timing diagrams, simpler implementations (in LD), and representation as given by IEC 61131:3 and IEC 60848. The transitions under which there are no ellipses are pit transitions. Two new symbols are proposed to extend the actions beyond the AES activity.

These timed actions are the following (one per row in Figure 5):

ON delay. It is coded as the output of a TON timer. In IEC 61131:3 it is represented with the qualifier D, and in IEC 60848, it is defined using symbol 24 (t_{on}/X^*), where X^* refers to the AES itself.

Time-limited action. This action is limited to either the duration of the AES or the preset time, whichever occurs first. It is coded as a TP timer in logical AND combination with the state of the AES X^* .

$$Out \leftarrow inst^{TP}.Q \text{ AND } X^*$$

where Out is a designated variable (typically an output), and $inst^{TP}.Q$ is the output of the TP timer instance used. In IEC 61131:3 it is represented with the qualifier L, and in IEC 60848 it is defined using symbol 25 ($t1/X^*$)

Timed pulse action. It is directly coded with the TP timer, and it is not limited to the AES duration. In IEC 61131:3 it is represented using the qualifier SL. The IEC 60848 does not directly consider this timed action in the general case where the output extends beyond the duration of the AES. To include it in a grafcet compliant with this standard, a Simultaneous Branch with Pit Transition (SBwPT) must be created, as shown in the third timed action in Figure 5. In the figure, pit transitions (transitions with no succeeding steps) are the ones under which there are no ellipses.

OFF delay. It is directly coded with a TOF timer. It is not covered by any of the standards; hence, its representation using GRAFCET requires an SBwPT, with an off-delay for the pit transition ($0s/X^*/t_{off}$ also expressed as $t_{off}\backslash X^*$).

Delayed and stored. It is coded as the output of a TON timer instance ($inst^{TON}.Q$) with the control signal:

$$Out \leftarrow (Out \text{ OR } inst^{TON}.Q) \text{ AND NOT } Reset$$

IEC 61131:3 represents it with DS. IEC 60848 does not consider this timed action but can be represented with an SBwPT (see Figure 5).

Stored and delayed. It would be coded with a TON timer whose control signal is the output of an RS bistable. This bistable is an element from the library if using an LD (ladder diagram) or FBD (function block diagram) language, or it must be coded manually if using IL (instruction list) or ST (structured text). IEC 61131:3 represents it with SD. IEC 60848 does not consider this timed action but can be represented with an SBwPT (see Figure 5).

A negative aspect of IEC 61131:3 regarding the last two timed actions (SD and DS) is that a *Reset* signal is needed for the timer deactivation, hence a SBwPT is also required for this standard. In addition, the coding of DS and SD is not straightforward, as Figure 5 shows.

In summary, except for the off delay and the handling of Reset in DS and SD, all possible timed actions can be represented as action qualifiers of the language SFC described in 61131:3. The timers TON, TOF and TP defined in 61131:3 as general function blocks and included in the library of most of the modern programming languages and environments are also introduced in this standard, which allows for their implementation in a straightforward or in a very simple way. In this respect, it represents an improvement over IEC 60848, which only implements TON. To extend the functionality of timed pulse action (TP) and off delay (TOF), it is proposed to prepend an “S” to the time-conditioned expression over the action box as in the rectangles in red in rows four and five of Figure 5.

With the proposed symbols, together with the timed action of the first row (t_{on}/X^*), IEC 60848 could directly represent these three timers (TON, TOF, and TP) already described in IEC 61131:3 as

$$\begin{aligned}
 t_{on} / X * (60848) &\leftrightarrow \text{timer TON (61131:3)} \\
 S \overline{t_p} / X * (\text{proposed for 60848}) &\leftrightarrow \text{timer TP (61131:3)} \\
 S \overline{X} * / t_{off} (\text{proposed for 60848}) &\leftrightarrow \text{timer TOF (61131:3)}
 \end{aligned}$$

and any embedded system could easily implement them as methods of a class. It is worth mentioning that most modern languages and environments (TIA Portal from Siemens, Codesys, Machine Expert from Schneider, and TwinCAT from Beckhoff) adhere to the specifications of IEC 61131:3, and therefore, they use these three timers.

Their pseudocodes are presented in Figure 6 for the methods (of the same class) implementing the three types of timers, as well as examples of use (see Figure 7) for the six timed actions of Figure 5.

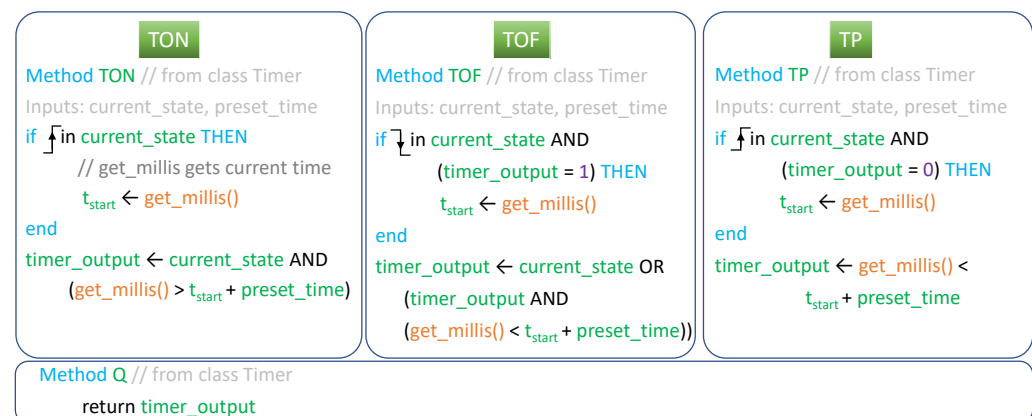


Figure 6. Pseudocode describing the operation of timing TON, TOF and TP, assuming that a class timer is defined in the software of an embedded system implementing GRAFCET.

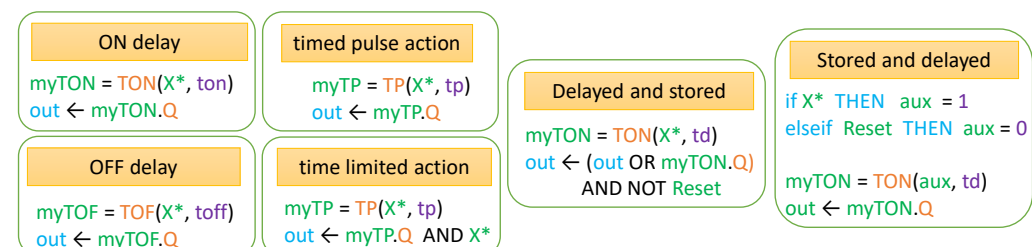


Figure 7. Examples of use for the different types of timed actions in Figure 5.

Regarding the SD and DS qualifiers, IEC 61131:3 introduced them but with significant drawbacks related to the representation of their Reset signal and their implementation. In most situations, these timed actions are not necessary, and when they are, they can be implemented with relatively simple modifications to the grafcet. For this reason, no changes are proposed to define them in standard IEC 60848.

3. Structure of the Project Implementing a Grafcet in an Embedded System

In this section, the structure of a project implementing a grafcet following the guidelines of IEC 60848 is presented. The code can be found in the repository [30] under the open-source CC BY 4.0 license, with a very basic project as a template and three more complex examples that are explained in Section 4. The repository includes instructions and videos to create the project for two different Integrated Development Environments (IDEs): STM32CubeIDE (together with STM32CubeMx) and Visual Studio Code + PlatformIO. The project structure is listed in Figure 8.

Each project has been tested on an STM32 microcontroller and an ESP32, using a different IDE for each one. So that the code can be reused for another microcontroller or IDE, a lower layer has been established that can be modified for another platform, thus allowing the upper layers to be reused. This corresponds to the blue layer in Figure 8.

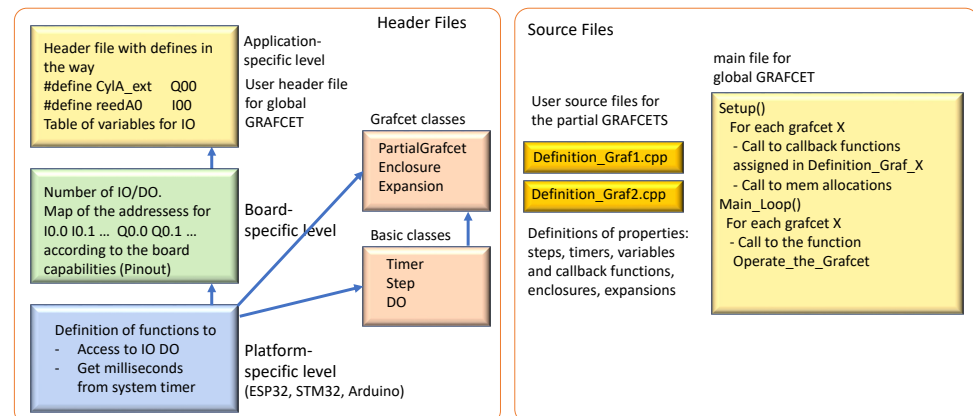


Figure 8. Structure of a project implementing the PG of a GG. The blocks in dark yellow are the ones corresponding to the PGs. The blocks in light yellow are the ones with the table of variables and the main file, both for the GG.

Details like the access to DIs and DOs, as well as the function to obtain the elapsed time, correspond to this level. As seen in Section 4.4, the projects have been compiled in seven platforms and launched in four of them.

Above is another layer (in green in the figure), which is supported by the platform-specific layer. It is specific to the connections of the board on which the microcontroller is mounted. Specifying the number and address of the available DIs and DOs corresponds to this level.

In order to use this code in another microcontroller/board or with another IDE, the content of the two header files which define these two levels must be adapted accordingly.

Based on these layers, several classes have been created (in salmon color) that should not be modified by the user. These classes are represented in Figure 9 and the main ones are the following:

PartialGrafcet. For the definition and operation of each PG. Two classes have been created that inherit from this one: expansions and enclosures.

DO. For the assignment of digital outputs. A class has not been created for digital inputs since their management is simpler, reduced to associating a `#define`.

Timer. For defining and consulting the timers.

Step. Created to facilitate the connections to enclosures and expansions.

An important aspect of this project is that there is a single class `PartialGrafcet`, of which as many instances are created as there are PGs in the user GG. Logically, the evolution, assignment of outputs, assignment of timers, etc., is different for each PG. This necessitates the use of callback functions in order to allow for different behaviors for each PG.

The right of Figure 8 shows the source files with the code implementing the PGs. The user must fill in a different source file for each PG, following the structure provided by the template file and the guidelines presented in this work. This file appears with the name `Definition_GrafcetX.cpp` in Figure 10, although it can be renamed. As shown in this figure, the user can choose the names of steps and timers and must program the transition

clearing (grafcet evolution), output assignment, timer definition and, where applicable, the definition of macrosteps, enclosures, forcing orders, and behavior in case of freezing. For each PG, the user must define a different operation function that contains these actions. The name of the specific function for each grafcet is assigned as one of the properties of the `PartialGrafcet` class as a callback function so that in each execution cycle this callback function is called to execute the specific tasks of the partial grafcet. In this way, in each PG, through its corresponding instance of the `PartialGrafcet` class, a different operational behavior can be defined (and executed).

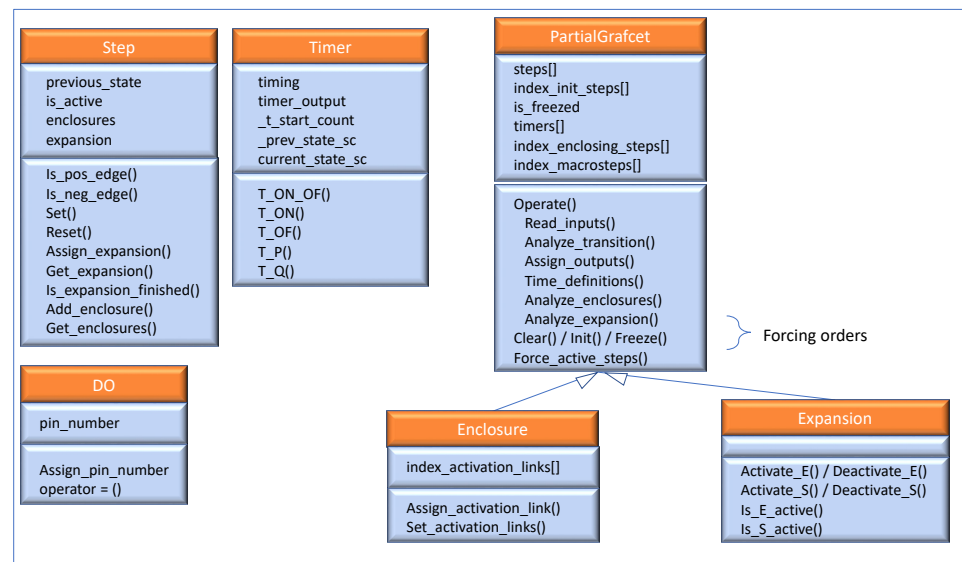


Figure 9. Classes used in the project.

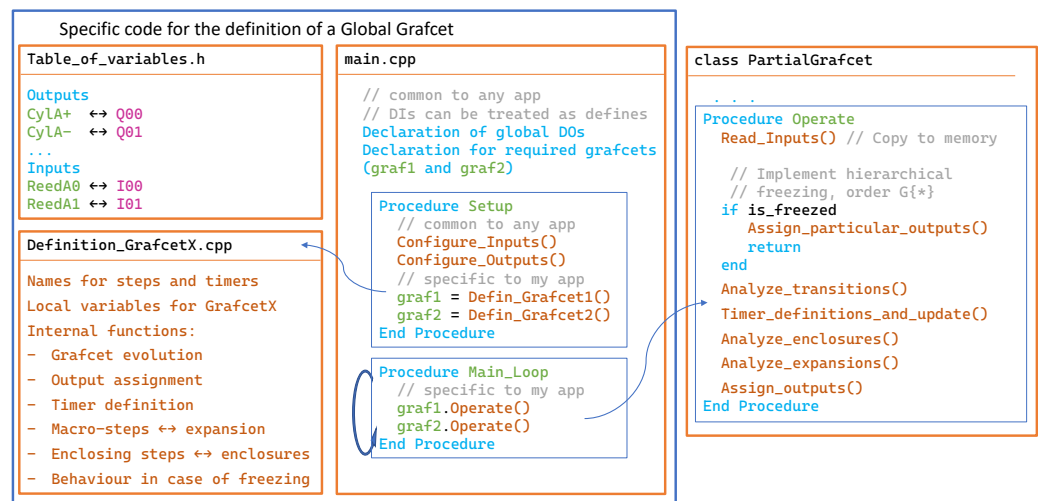


Figure 10. Structure of the project with the main file and the operation function that is called cyclically.

4. Global Grafkets Implemented with the Proposed Tool

Below are three examples covering all the items collected by the IEC 60848 standard. They are presented in order of complexity.

4.1. GG with Macrosteps

A macrostep serves as a placeholder of a part of the specification and allows for the composition and decomposition of a grafcet. It is represented graphically by the step symbol with two additional horizontal lines at the top and bottom and the prefix M in its name. Because of its placeholder function, no action should be associated to a macrostep.

In Figure 11, a simple global grafcet with a start/stop switch (swON_OFF) containing two macrosteps is shown. The corresponding expansions are developed in different PGs, which are reduced to the reciprocating movement of two cylinders, one controlled by a bistable solenoid valve and the other by a monostable one. On the right side, the pseudocode for a fragment of the evolution of the main grafcet's task is displayed.

This example uses the following resources:

- Timers TON.
- Distinction between monostable and bistable solenoid valves.
- Management of macrostep expansion: initialization, query of its completion, and deactivation.

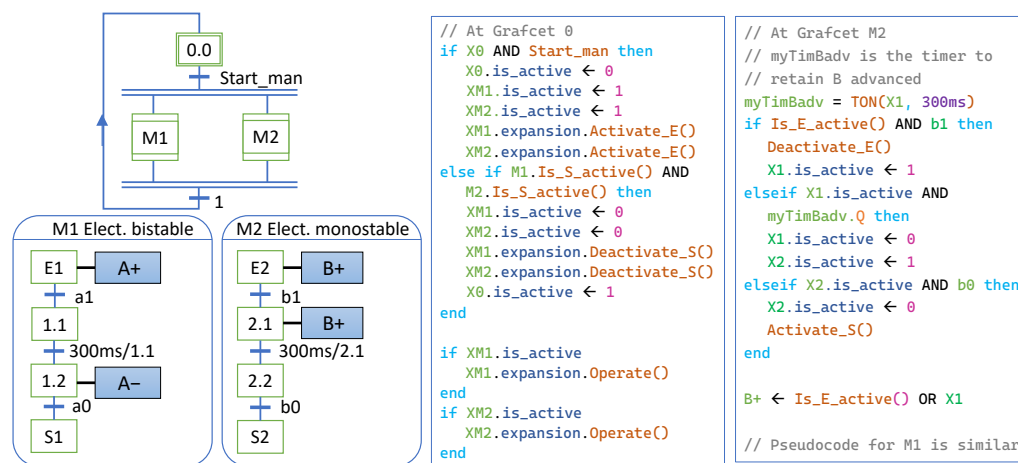


Figure 11. Example of GG with macrosteps. On the right, an excerpt of the pseudocode managing the interaction between PGs.

As previously mentioned, a class Expansion has been created, inherited from PartialGrafcet, with the methods to activate/deactivate the first and last steps of the expansion and to consult them. On the other hand, the Step class contains properties and methods to link a stage (macro-stage) with an instance of the Expansion class. In grafcet 0, Analyze_Expansions is called to determine which steps are macrosteps, the associated expansion is detected using Get_expansion, and if the macrostep is active, the Operate method of the expansion is called to analyze transitions, define timers and assign outputs.

4.2. Hierarchical Control with Forcing Orders

The GG is shown in Figure 12. The main grafcet from the functional point of view is grafcet 2, in which a loop of operations with coordinated cylinders is executed. Grafcet 0, hierarchically superior, controls its start-up and allows for a stop at the end of the cycle (Stop EoC). This grafcet, by forcing the initialization of PG1, also controls the stop at the end of the iteration (Stop_EoI). The stop buttons should be normally closed (NC) for safety so that they stop the process in case of wire breakage or accidental disconnection of the button. To the right, the pseudocode of the fragments that describe the specific actions to be carried out to manage the forced commands is shown: in PG0, the freezing of PG2 and the initialization of PG1 are forced; in PG2, the timed pulse action and how its own freezing is performed are described. The PG1 as well as other transition analyses in PG0 and PG2 have not been shown for the sake of space.

This example uses the following resources:

- Timers and counters. Timing is equivalent to SL of the IEC 61131:3 standard as X2.0 action and is coded as a TP timer commanding the output H. As mentioned, a new symbol is proposed (third row of Figure 5) to represent a timed pulse action.

- Distinction between monostable and bistable solenoid valves.
- Use of NC contacts for stop operations.
- Transitions dependent on the state of steps of another PG.
- Action on activation and action on deactivation.
- Source steps.
- Forcing orders of PGs to different situations (explicit and freezing/current).

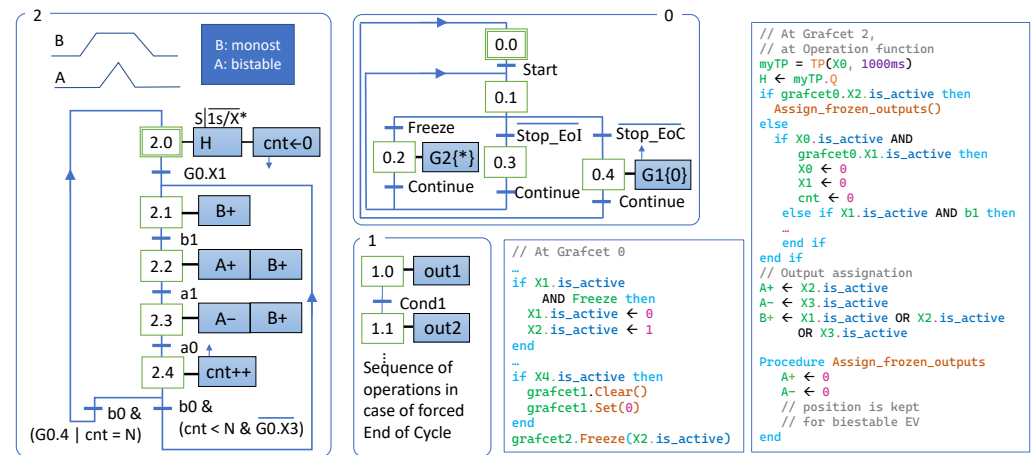


Figure 12. Example of GG with forcing orders. On the right, an excerpt of the pseudocode managing the interaction between PGs.

In a GG with several PGs (that are not expansions or enclosures), all PGs must be launched simultaneously. However, in this example, PG1 does not start from any active step initially and will wait to be forcibly activated from step 4 of PG0. All PGs operate simultaneously, although in this example, PG0 controls the operation of PG1 and PG2 through forcing orders, and PG2 also checks the status of some steps in PG0.

4.3. Hierarchical Control with Enclosing Steps

Enclosing steps offer a way of implementing a hierarchical structure between autonomous PGs. The activation of an enclosing step of a higher-level PG simultaneously activates specific steps of a lower-level PG, referred to as PG of enclosing steps or enclosed PG. It is similar to macrostep expansions, which are mainly intended to improve GRAFCET documentation. However, unlike macrosteps, this feature has a cyclic nature and is therefore designed to contain the description of an autonomous operation.

As depicted in Figure 13, the main grafcet from a functional perspective is PG1, which comprises two connected grafcets. A typical loop of operations for a rotary table is executed within them. In the first sector, there is a piece feeder; in the last one, the piece is extracted; and in the intermediate ones, it is machined. Grafcet 0, hierarchically superior, controls its start-up and stop, which must be implemented with an NC contact. There is a counter that counts the number of pieces manufactured in general, which can be reset.

This example utilizes the following resources:

- Counters.
- Use of NC contacts for stop operations.
- Action on activation and action on deactivation.
- Action on event. In the IEC 60848 standard and in the consulted bibliography, its condition always appears as an edge although, as shown in this case, it is not necessary.
- Source transitions and pit transitions.

- Management of enclosing steps (in this case, initial enclosing step): activation of the enclosure with the rising edge of the enclosing step, either by initializing the enclosure, or activating its steps with activation links; and operation of the enclosure.
- Assembly line structure with several simultaneous execution points in a sequential structure. This is an alternative version to simultaneous processes carried out through divergence–convergence in AND. They require a synchronization signal, generally in the form of an edge, which in this case is the end of the rotation of a rotary table.

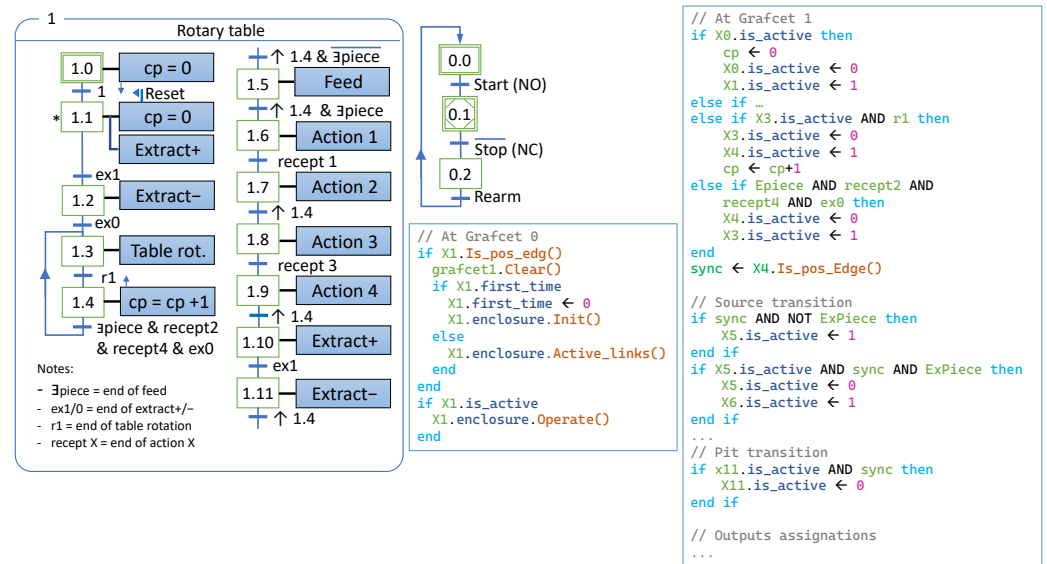


Figure 13. Example of GG with enclosing step and its corresponding enclosures (in this case only one). At the left, excerpt of the pseudocode managing the interaction between PGs.

With regard to the implementation of the enclosures, the mechanism is similar to that of macrosteps, although with important differences:

- There is no single starting step; there may be several. These are the steps marked as initial in the first call to the enclosure, or those marked with activation links in subsequent calls.
- In an expansion, there is an exit step after which the expansion is immediately exited. In an enclosure, there is no explicit exit step; it is exited when the transition following the encapsulating step is crossed. In the first case, the exit depends on the completion of the expansion, whereas in the second, it depends on the evolution of the invoking grafcet in which the encapsulating step resides.
- Several enclosures can be linked to only one enclosing step. This is not the case with macrosteps, which can only be associated with an expansion.

With these examples all the characteristics described in the standard have been reviewed, and Table 3 lists the location where the advanced resources are explained or used:

- Basic elements and structures: steps, transitions, continuous actions, simultaneous subprocesses, selection of subprocesses, and digital inputs/outputs.
- Input events and internal events such as rising/falling edge detection of binary variables.
- Stored actions: allocation of values to variables, action on activation, action on deactivation, and action on event.
- Timers in the format t1/sc/t2, as specified in standard IEC 60848, and used in transitions. When used in conditioned actions, they have been assimilated to one of the six timed actions described in IEC 61131:3. Although they are not described in the IEC 60848, TON-, TOF-, and TP-type timers have also been implemented, which appear at

lower levels of description (such as that included in IEC 61131:3), but that can serve as a connecting bridge, if necessary, with already established programming languages.

- Hierarchical structures: enclosing steps + enclosures, macrosteps + expansions, and forcing orders of a grafcet to different situations (explicit, freeze, empty, and initial).

Table 3. List of explained resources and the figure where they are found. Examples in C++ for all of the resources are available in [30].

Resources	Structure GRAFCET	Structure (Code/Pseudocode)	Example of Use in GRAFCET	Example of Pseudocode
Timed actions	Figure 5		Figures 5 and 12	Figure 7
Timers TON TOF TP		Figure 6		Figure 7
Timed transitions			Figures 11 and 12	Figures 11 and 12
Rising edge in transition			Figures 11 and 13	Figures 11 and 13
Continuous actions	Figure 1	Figure 1	Figure 12	Figure 12
Stored actions	Figure 2	Figure 2	Figure 12	Figure 12
Counters			Figure 12	Figures 12 and 13
Macros and expansions			Figure 11	Figure 11
Forcing grafkets: freeze/init			Figure 12	Figure 12
Encapsulated grafkets			Figure 13	Figure 13
Source and pit transitions	Figure 5		Figure 13	Figure 13
Source and pit steps			Figure 11	Figure 11

4.4. Implementation on Different Microcontrollers and IDEs

The project is intended to be flashed in a 32-bit microcontroller, initially on an STM32 and ESP32. Several attempts have been made to flash the code in 8-bit microcontrollers (PIC, 8051, Arduino), but excessive inconveniences arise with regard to its programming in C++, or its ROM capacity. Finally, the project has been uploaded and launched in an STM32F103RB, STM32F411RE, ESP32 (WROOM-32) and Arduino DUE. It has also been compiled, but not launched, in a microcontroller NXP i.MXTR1062 (board Teensy 4.1) and in a Renesas RA4M1 (Arduino UNO R4 Minima). For these two boards, since the allocation of the array of steps and timers is dynamic, the required RAM values of the table could not be observed and they have been assumed equal to ESP32 or Arduino DUE.

In Table 4, the RAM and ROM memory required for the template project and for the other three projects are shown for the different platforms tested. Another row has been included corresponding to a project with 100 steps and 25 timers to check the low requirements for RAM and ROM in a medium-size project. As can be seen, the memory requirements remain well below the microcontroller's capacity, meaning that it can host global grafkets of this size or significantly larger. In this case, the limitation is more likely to be determined by the number of available inputs and outputs.

Table 4. RAM and Flash memory usage (in bytes and percentage) for different projects and microcontrollers.

Memory	Project	STM32F411RE	STM32F103RB	ESP32	Arduino DUE	Teensy 4.1	Renesas
RAM	Template	316 (0.24%)	316 (1.54%)	336 (0.06%)	336 (0.34%)	336 (0.06%)	336 (1.02%)
RAM	Forcing	956 (0.73%)	988 (4.82%)	1048 (0.20%)	1048 (1.1%)	1048 (0.20%)	1048 (3.20%)
RAM	Macrosteps	956 (0.73%)	988 (4.82%)	1048 (0.20%)	1048 (1.1%)	1048 (0.20%)	1048 (3.20%)
RAM	Enclosures	844 (0.64%)	892 (4.12%)	892 (0.17%)	892 (0.90%)	892 (0.17%)	892 (2.72%)
RAM	100S-25T	2536 (1.93%)	2544 (12.4%)	2564 (0.48%)	2564 (2.60%)	2564 (0.49%)	2564 (7.82%)
ROM	Template	12,680 (2.4%)	10,444 (8.0%)	255,737 (19.5%)	13,612 (2.6%)	12,356 (0.15%)	34,812 (13.3%)
ROM	Forcing	13,892 (2.6%)	11,648 (8.9%)	257,237 (19.6%)	14,636 (2.8%)	13,572 (0.16%)	35,840 (13.7%)
ROM	Macrosteps	16,720 (3.2%)	14,440 (11.0%)	257,853 (19.7%)	15,072 (2.9%)	14,140 (0.17%)	36,252 (13.8%)
ROM	Enclosures	16,824 (3.2%)	14,540 (11.1%)	257,617 (19.7%)	14,904 (2.8%)	14,092 (0.17%)	36,076 (13.8%)
RAM	100S-25T	19,740 (3.7%)	16,820 (12.9%)	259,221 (19.8%)	17,472 (3.3%)	19,236 (0.23%)	38,712 (14.8%)

Note: For the Teensy 4.1 platform, RAM usage corresponds to RAM1 (512 KB), while program memory is stored in external QSPI flash (8 MB).

It is also observed that all the evaluated platforms provide sufficient RAM and ROM capacity for small- and medium-sized embedded applications without memory limitations (RAM usage < 4% for most of them). Although ESP32 and Renesas exhibit a higher baseline ROM usage due to the mandatory inclusion of core libraries and runtime components, regardless of the project's complexity, the required ROM does not increase significantly (less than 1.5%) as the application incorporates more transitions, steps and timers.

Table 5 presents the cycle time for the first four platforms. For the projects related to macrosteps and enclosures, a distinction is made between the case when enclosures or macrosteps are active and when they are not.

Table 5. Cycle time (in μs) for different projects and microcontrollers.

Project	STM32F411	STM32F103	ESP32	Arduino DUE
Template	14 μs	23 μs	9.6 μs	80 μs
Forcing	53 μs	83 μs	15 μs	340 μs
Macrosteps	75–26 μs	38–108 μs	9–18 μs	102–411 μs
Enclosures	16–63 μs	27–147 μs	8.4–20 μs	88–553 μs

It can be observed that, except for the Arduino DUE, the cycle time is on the order of 0.1 ms, remaining in all cases well below the cycle times of an industrial PLC, which are typically around one millisecond.

5. Discussion and Conclusions

5.1. Discussion

5.1.1. Modularity and Hierarchy

From the authors' perspective, IEC 60848 has not been very ambitious in implementing modularity and hierarchy, and it is very limited in terms of code reusability. This limitation arises from the use of absolute/global references in step definitions, which prevents the declaration of modular structures needed to build large-scale projects. Using global references for all grafkets, regardless of their hierarchical level, can be likened to relying solely on global variables in a microcontroller or microprocessor program, severely restricting the use of functions and structured programming.

5.1.2. Enclosing Steps

Another limitation of the standard is the lack of explanation regarding the utility of macrosteps and enclosing steps as means of achieving hierarchy in grafkets.

The expansion of macrosteps define a start and an end and are suitable for separating a part of the grafket for documentation purposes and to offer a clearer vision of the solution.

The enclosed steps naturally return to the beginning and are more related to hierarchy. If an enclosing step is active, then the enclosure is executed. One possible use case is found during the normal operation of a process or station, which is controlled by a hierarchically superior grafket that determines whether the system is in operating or stop mode. If the higher-level grafket is not in the normal operating stage, the operational grafket would be deactivated.

In this case, but also in any other, if the enclosing step is deactivated, it will be necessary to selectively reset the outputs activated during the execution of the enclosed steps. This requires either manually deactivating each output, or knowing how they were activated, either by assignation (continuous action, symbol 20), or allocation (stored action, symbol 26), in order to forcibly reset the stored actions. In any case, it is necessary to understand the details of a lower-level grafket, which goes against the principle of modular code design.

To avoid this lack of modularity, it is forced using FBs, and the programmer should be responsible for performing these resetting task when defining the FB.

5.1.3. Preference for C++

The pseudocode presented throughout this work allows for implementation in any programming language; however, the projects provided in [30] are written in C++. Compared to another language, such as the popular Python, C++ offers the following advantages:

Lower resource consumption. Python requires an interpreter, which makes it heavy and not very suitable for embedded devices with limited resources.

Optimal in code and speed. C++ is a compiled, mid-level programming language that combines high-level abstraction with low-level control, allowing for highly efficient codes in terms of both execution speed and memory usage. As seen in Table 5, the cycle times are less than 0.1 ms, lower than those of an industrial PLC. Regarding RAM requirements, a Step object typically occupies 20 bytes and a Timer object only 16 bytes, resulting in a negligible RAM overhead that imposes virtually no limitation on project size.

Direct hardware access. C++ enables low-level control through assignment and reading of registers, whereas Python code requires external libraries that are not always stable in industrial environments.

Error occurrence. Being a strongly typed language, C++ does not allow for compilation of code with typing errors, which may go unnoticed in Python and only appear at runtime, potentially causing unpredictable errors during system operation.

Debugging capabilities. Most C++ IDEs offer the possibility of detecting errors and verifying their operation through advanced debugging tools, which allow for the setting of breakpoints (conditional or unconditional), step-by-step execution, inspection of registers and variables, modification of variable states, etc. This facilitates project commissioning, monitoring, and error detection.

Due to the structured nature of the C++ language, the resulting code from implementing different examples ends up being very similar to the structured language ST, except for specific editing rules, which, in the case of ST, are closer to PASCAL. Consequently, the authors consider that this work contributes to the design of a systematic method for translating the GRAFCET model defined in standard IEC 60848 into a structured language. However, some shortcomings in the standard have also been discovered that need to be specified for such translation to be complete and unambiguous.

5.2. Conclusions

In the Introduction, different metamodels have been referenced that represent in declarative languages the rules and concepts of GRAFCET. With the IEC 61131:3 standard, there is an attempt to define the semantics of various textual and graphical languages systematically to capture the evolution and operation of a grafcet, making it closer to programming in modern PLC languages. A clear example can be seen in timed actions, whose functionality was very limited in IEC 60848, as they were restricted to the scope of a single step.

Thus, there exists a gap between the two standards. This work aims to identify these deficits and sources of vagueness in order to provide solutions for them. According to the guidelines of the IEC 60848 standard, we have defined different hierarchical structures (PGs, enclosures, and macrostep expansions) in a more modular manner, employing local variables for step definitions, counters, and timers within each of these structures. The invoking mechanism and completion acknowledgment of FBs (an essential resource for hierarchy management) have been clearly described.

According to this purpose, the first objective of this paper has been to unequivocally explain how to implement the structures proposed by standard IEC 60848 into any programming language. Table 1 show the contributions of this work with regard to this first objective.

The second objective is providing the guidelines to transcribe in C++ all the semantics of GRAFCET indicated in the IEC 60848 standard, together with the improvements introduced in IEC 61131:3, into an embedded system so that it can control and/or supervise several processes simultaneously, without the need to use real-time software or interrupts. Table 2 show the contributions of this work with regard to previous ones.

The project has been tested in debugging mode for the STM32F103 and STM32F411 microcontrollers using STM32CubeIDE, which allows for free manipulation of the virtual digital inputs (the ones of the PIM) for this board. It has also been tested on the ESP32 microcontroller using custom-made electronic circuits designed to graphically display the evolution of the grafcet. The connection of these boards allows for digital inputs to be actuated at will.

This research successfully addresses the inherent ambiguities, vagueness, inaccuracies, and non-modular limitations within the IEC 60848:2013 standard that previously hindered an unequivocal programmatic translation to embedded systems. By establishing rigorous interpretations and extensions, this work provides clear guidelines, pseudocode and structures for implementing advanced GRAFCET resources, such as timers, stored actions, macrosteps, enclosures, and forcing orders—which are typically overlooked in the existing literature.

The practical viability of these implementations was validated through experimental evaluation across various 32-bit microcontroller architectures, specifically the STM32 and ESP32 platforms. The empirical results demonstrate that the proposed single-threaded sequential execution engine delivers superior computing speeds, with cycle times consistently hovering around 0.1 ms or lower, bypassing the need for complex operating systems or RTOS. Moreover, the implementation exhibits exceptional memory efficiency under scaled stress-testing conditions (100 steps and 25 timers): less than 2% of the RAM for STM32F411RE and ESP32-Wroom32 and less than a 1.5% increase in ROM when implementing the analysis of 100 transitions and the definition of 25 timers. This proves its scalability for complex discrete-event applications.

While traditional PLCs remain the logical and preferred standard for purely industrial processes, this framework establishes microcontroller-based implementations as a robust, ultra-lightweight, and cost-effective alternative. It offers a highly suitable solution for specialized embedded environments, custom IoT devices, and cost-sensitive automation systems where hardware versatility, compactness, and high performance are essential.

5.3. Using RTOS in Future Applications

As mentioned earlier, a global grafcet, like the ones presented as examples, can be coded in a single thread using the proposed tool. However, it has not been tested in other situations that, e.g., include serial communication tools (SPI, I2C, and WiFi). To this end, a future development approach will involve including RTOS instructions (and libraries) to implement this communication robustly while maintaining good GRAFCET execution performance. This could be the case of requiring input/output expander cards to increase the number of DIs and DOs.

Beyond RTOS-based execution and communication-enabled architectures, future work will investigate the integration of the proposed GRAFCET framework with real-time prediction and visualization, extending it toward a data-driven decision support framework for embedded industrial systems.

Future work will also address additional industrial reliability and safety requirements, including emergency-stop handling, fail-safe output strategies, watchdog mechanisms, input debouncing, brownout management, and safe restart procedures after power failures or communication loss, further improving the robustness of the proposed method.

Author Contributions: A.G.G.-R.: Conceptualization, methodology, writing—original draft preparation, software, validation, formal analysis. E.E.-E.: Supervision, validation, review, and formal analysis. J.V.M.-D: Supervision, review, and editing. P.J.C.-P.: Methodology, supervision, and software. J.C.-B.: Supervision, review, and software. All authors have read and agreed to the published version of the manuscript

Funding: This work has been funded by the University of Jaen through the 2025 Research and Knowledge Transfer Plan (Program R.1. UJA Investiga, Project Ref. PID2024-161452OB-I00).

Informed Consent Statement: Not applicable.

Data Availability Statement: The code containing the examples that implement the GGs presented in Section 4, as well as the template for developing any other GG, can be found in [30].

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AES	Action-Execution Step
DS	Delayed and Stored
FB	Function Block
GG	Global Grafcet
LD	Ladder
PG	Partial Grafcet
SBwPT	Simultaneous Branch with Pit Transition
SD	Stored and Delayed
TS	Transient Step

References

- Schmuck, A.K.; Moor, T.; Majumdar, R. On the Relation Between Reactive Synthesis and Supervisory Control of Input/Output Behaviours. *IFAC-PapersOnLine* **2018**, *51*, 31–38. <https://doi.org/https://doi.org/10.1016/j.ifacol.2018.06.275>.
- Partovi, A.; Lin, H. Reactive Supervisory Control of Open Discrete Event Systems. In Proceedings of the 2019 IEEE 58th Conference on Decision and Control (CDC), Nice, France, 11–13 December 2019; pp. 1056–1061. <https://doi.org/10.1109/CDC40024.2019.9030253>.
- Baeten, J.; van Beek, B.; van Hulst, A.; Markovski, J. A process algebra for supervisory coordination. In Proceedings of the Electronic Proceedings in Theoretical Computer Science, Reykjavik, Iceland, 9 June 2011; Volume 60, pp. 36–55. <https://doi.org/10.4204/EPTCS.60.3>.
- Zaytoon, J. On the recent advances in Grafcet. *Prod. Plan. Control.* **2002**, *13*, 86–100. <https://doi.org/10.1080/09537280110063735a>.
- IEC 60848:2019; General Principles of Reliability for Structures. International Organization for Standardization: Geneva, Switzerland, 2019.
- Barbieri, G.; Gutierrez, D.A. A GEMMA-GRAFCET Methodology to enable Digital Twin based on Real-Time Coupling. *Procedia Comput. Sci.* **2021**, *180*, 13–23. <https://doi.org/10.1016/j.procs.2021.01.122>.
- Mross, R.; Schnakenbeck, A.; Völker, M.; Fay, A.; Kowalewski, S. Unambiguous Interpretation of IEC 60848 GRAFCET based on a Literature Review. In Proceedings of the 2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA), Sinaia, Romania, 12–15 September 2023; pp. 1–9. <https://doi.org/10.1109/ETFA54631.2023.10275504>.

8. Provost, J.; Roussel, J.M.; Faure, J.M. Translating Grafcet specifications into Mealy machines for conformance test purposes. *Control. Eng. Pract.* **2011**, *19*, 947–957. <https://doi.org/10.1016/j.conengprac.2010.10.001>.
9. Le Parc, P.; L'Her, D.; Scharbarg, J.L.; Marcé, L. Grafcet revisited with a synchronous data-flow language. *IEEE Trans. Syst. Man Cybern. Part A Syst. Hum.* **1999**, *29*, 284–293. <https://doi.org/10.1109/3468.759279>.
10. Provost, J.; Roussel, J.M.; Faure, J.M. A formal semantics for Grafcet specifications. In Proceedings of the 2011 IEEE International Conference on Automation Science and Engineering, Trieste, Italy, 24–27 August 2011; pp. 488–494. <https://doi.org/10.1109/CASE.2011.6042457>.
11. Julius, R.; Schürenberg, M.; Schumacher, F.; Fay, A. Transformation of GRAFCET to PLC code including hierarchical structures. *Control. Eng. Pract.* **2017**, *64*, 173–194. <https://doi.org/10.1016/j.conengprac.2017.03.012>.
12. Julius, R.; Trenner, T.; Neidig, J.; Fay, A. A model-driven approach for transforming GRAFCET specification into PLC code including hierarchical structures. *IFAC-PapersOnLine* **2019**, *52*, 1767–1772. <https://doi.org/10.1016/j.ifacol.2019.11.457>.
13. Castillo, J.M.; Barbieri, G.; Mejia, A.; Hernandez, J.D.; Garces, K. A GEMMA-GRAFCET Generator for the Automation Software of Smart Manufacturing Systems. *Machines* **2021**, *9*, 232. <https://doi.org/10.3390/machines9100232>.
14. Couffin, F.; Lamperiere, S.; Faure, J.M. Contribution to the Grafcet formalisation: A static meta-model proposition. *J. Eur. Des Syst. Autom.* **1997**, *31*, 645–667.
15. Julius, R.; Trenner, T.; Fay, A.; Neidig, J.; Hoang, X.L. A meta-model based environment for GRAFCET specifications. In Proceedings of the 2019 IEEE International Systems Conference (SysCon), Orlando, FL, USA, 8–11 April 2019; pp. 1–7. <https://doi.org/10.1109/SYSCON.2019.8836959>.
16. Mross, R.; Schnakenbeck, A.; Volker, M.; Fay, A.; Kowalewski, S. Transformation of GRAFCET Into GAL for Verification Purposes Based on a Detailed Meta-Model. *IEEE Access* **2022**, *10*, 125652–125665. <https://doi.org/10.1109/ACCESS.2022.3225736>.
17. Schumacher, F.; Fay, A. Formal representation of GRAFCET to automatically generate control code. *Control Eng. Pract.* **2014**, *33*, 84–93. <https://doi.org/10.1016/j.conengprac.2014.09.008>.
18. IEC 61131-3:2013; Programmable Controllers—Part 3: Programming Languages. International Organization for Standardization: Geneva, Switzerland, 2013.
19. Witsch, D.; Vogel-Heuser, B. Close integration between UML and IEC 61131-3: New possibilities through object-oriented extensions. In Proceedings of the 2009 IEEE Conference on Emerging Technologies & Factory Automation, Palma de Mallorca, Spain, 22–25 September 2009; pp. 1–6. <https://doi.org/10.1109/ETFA.2009.5347155>.
20. Wall, D.; Dixon, J.; Fischer, J. *Guidelines for Usage of Object-Oriented Programming*; Technical Report, PLCopen for Efficiency in Automation; PLCopen, Gorinchem, The Netherlands, 2021.
21. Bauer, N.; Huuck, R.; Lukoschus, B.; Engell, S. A unifying semantics for sequential function charts. In *Integration of Software Specification Techniques for Applications in Engineering*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2004; Volume 3147, pp. 400–418. https://doi.org/10.1007/978-3-540-27863-4_22.
22. Gonzalez-Rodriguez, A.G.; Vadillo-Marin, J.; Gonzalez-Rodriguez, A.; Ruiz-Bailen, M. A Detailed Description for a Volume-Controlled Mechanical Ventilator Operated with Solenoid Valves. 2020. Available online: https://mecatrond.es/Cycled_ventilator.pdf (accessed on 20 June 2026).
23. Comlan, M.; Delfieu, D.; Assogba, N. Grafcet to Arduino: Edit and upload Grafcets on an Arduino boards. In Proceedings of the 2021 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), Mauritius, Mauritius, 7–8 October 2021; pp. 1–6. <https://doi.org/10.1109/ICECCME52200.2021.9590933>.
24. Bayó-Puxan, O.; Rafecas-Sabaté, J.; Gomis-Bellmunt, O.; Bergas-Jané, J. A GRAFCET-compiler methodology for C-programmed microcontrollers. *Assem. Autom.* **2008**, *28*, 55–60. <https://doi.org/10.1108/01445150810849019>.
25. Nzebop Ndenoka, G.; Simeu, E.; Alhakim, R. Efficient controller synthesis of multienergy systems for autonomous domestic water supply. *Rev. Afr. Rech. Inform. Math. Appl.* **2017**, *24*, 65–68. <https://doi.org/10.46298/arima.1458>.
26. Chmiel, M.; Mocha, J.; Hryniewicz, E.; Polok, D. About implementation of IEC 61131-3 IL operators in standard microcontrollers. *Ifac Proc. Vol.* **2013**, *46*, 144–149. <https://doi.org/10.3182/20130925-3-CZ-3023.00040>.
27. Mallet, F.; Gaffe, D.; Boeri, F. Concurrent control systems: from Grafcet to VHDL. In Proceedings of the 26th Euromicro Conference. EUROMICRO 2000. Informatics: Inventing the Future, Maastricht, The Netherlands, 5–7 September 2000; Volume 1, pp. 230–234. <https://doi.org/10.1109/EURMIC.2000.874637>.
28. Wiesmayr, B.; Zoitl, A.; Miguel-Escrig, O.; Romero-Perez, J.A. Distributed implementation of hierarchical Grafcets through IEC 61499. In Proceedings of the 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vasteras, Sweden, 7–10 September 2021. <https://doi.org/10.1109/ETFA45728.2021.9613251>.

29. Schnakenbeck, A.; Mroß, R.; Völker, M.; Kowalewski, S.; Fay, A. Structural Analysis of GRAFCET Control Specifications. In Proceedings of the 2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA), Sinaia, Romania, 12–15 September 2023; pp. 1–9. <https://doi.org/10.1109/ETFA54631.2023.10275713>.
30. Gonzalez, A.; Ottaviano, E.; Rea, P.; Muñoz-Diez, J.V. Implementing a GRAFCET following Standard IEC-60848 in an ESP32. *Mendeley Data* 2024. <https://doi.org/10.17632/3dh59pp7t3>.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.