Distributed Implementation of Hierarchical Grafcets through IEC 61499

Bianca Wiesmayr*, Alois Zoitl*†, *Member, IEEE*
†*CDL VaSiCS*
**LIT CPS Lab, Johannes Kepler University Linz*
Linz, Austria
{bianca.wiesmayr, alois.zoitl}@jku.at

Oscar Miguel-Escrig, Julio-Ariel Romero-Pérez
Department of Systems Engineering and Design
Universitat Jaume I
Castellon, Spain
{omiguel, romeroj}@uji.es

***Abstract*—The control software of manufacturing systems has to interact with the mechanical and electrical machine parts, hence, their engineering is closely interrelated. A high-level model of this system can provide an abstract view on the system's behavior and is therefore beneficial for any engineering discipline. The language GRAFCET was developed as a domain-specific tool for specifying the operation of a production system. GRAFCET models can be structured hierarchically to regulate the behavior of a subsystem with forcing orders and enclosing steps. Based on a Grafcet, control software can be implemented in the executable modeling language IEC 61499. In this paper, we describe a step-by-step approach for implementing hierarchical features of GRAFCET with IEC 61499 models. The resulting application is well-structured and thus maintainable.**

***Index Terms*—IEC 60848, GRAFCET, IEC 61499, modeling control systems**

I. Introduction

The design of production systems typically involves various engineering disciplines, hence, a common understanding of the system functionality is essential. Several modeling languages have been developed to unambiguously capture the operation of an automation system without focusing on implementation details. Examples for such languages are Petri Nets as well as GRAFCET, a domain-specific modeling language (DSML) for automation [1], which is based on Petri Nets. The features provided by GRAFCET enable developers to model both simple and more complex systems. Grafcet models are comprised of steps and transitions and thus well understandable [2]. GRAFCET provides various tools for describing complex behaviors by means of structured and hierarchical models.

A systematic approach which is used in practice for modelling hierarchic control applications is based on GEMMA (*Guide d'Etude des Modes de Marche et d'Arrêt*) [3]. This guide describes a systematic approach to develop hierarchical models for automation systems by identifying their relevant operation modes based on 16 predefined modes, which are classified into three groups: production procedures, stop procedures, and failure procedures. The individual procedures are then modeled and their coordination results in the desired overall behavior. In the domain of packaging machines, a GEMMA-based approach is the so-called PackML state machine (Packaging Machine Language). It is a recommended state model for handling different operation modes [4].

In GRAFCET, operation modes are frequently modeled as forcing orders. The resulting models are structured hierarchically. When implementing such models as distributed control software, corresponding language patterns in IEC 61499 [5] are required. We aim at providing a standardized approach for translating Grafcet models into control applications based on IEC 61499, including the Grafcet's structuring mechanisms. In this paper, we therefore extend our methodology from [6], where we compared the two modeling languages and outlined the basic methodology for translation. We provided numerous utile patterns for common language features. Structuring features were not covered in our previous work. In this paper, we cover (1) the forcing order of steps in GRAFCET, (2) enclosing steps, and (3) macro-steps. These language features are described in detail in Section III, while their implementation in IEC 61499 is covered in Section VI. We will illustrate our approach based on a practical example (Section VII).

This work has been supported by research project 18I411-UJI-B2018-39 from Universitat Jaume I and by CEICE grant number ACIF/2018/244.

II. Related Work

The control software derived from GRAFCET models is executed on Programmable Logic Controllers (PLCs). In a distributed system, the control software is split across several PLCs. IEC 61499 [5] is a well-suited target language for implementing distributed control software, because it defines modular units (Function Blocks, FBs) that can be interconnected to form an application model. Each FB is then assigned to the device that executes it. IEC 61499-applications are thus decoupled from the hardware configuration, which is a major advantage compared to the languages defined in the preceding standard IEC 61131-3 [7]. Like GRAFCET, IEC 61499 supports hierarchical structuring, i.e., a production system can be controlled by a composition of several smaller models. A structure that resembles the mechatronic architecture of a production system is typically preferred [8].

Several works treat the translation of Grafcets to languages defined in IEC 61131-3 (e.g., [2], [9]). In [2], an extensive approach is available for Structured Text that also hierarchically structures the generated code to ensure readability. The functionality of a partial Grafcet is implemented within an FB as Structured Text (ST) and forcing orders are executed by a specific method. In IEC 61131-3, FBs are program units that can call each other or can be interconnected graphically in an FB Diagram [10]. Another work proposes to normalize a Grafcet before implementing it in a Sequential Function Chart (SFC) [9]. This approach results in large SFCs that are difficult to comprehend [2]. SFC has limited expressiveness and does not support hierarchical elements.

For distributed control software, IEC 61499 defines models that can be mapped to any number of devices. As the Application is not affected by a new hardware mapping, individual devices need not be considered in a high-level model like Grafcet. However, the implementation of hierarchical GRAFCET features in IEC 61499 has not been discussed yet. When translating hierarchical features from GRAFCET to IEC 61499, existing guidelines for structuring the software should be considered. For example, applications can be structured strictly hierarchically where a single component orchestrates its subordinates [11]. In a variation of this pattern [12], loosely coupled application parts communicate as peers. Implementing the operation modes of a machine within an FB was analysed based on the PackML state machine in [4].

III. Structuring models hierarchically

Abstraction hides the detailed implementation of a component. For example, several blocks can be represented as a single component. This provides several benefits for any modeling language: it improves comprehensibility and reduces the risk of errors [13]. It furthermore fosters the reuse of submodels, especially if design patterns are taken into account [14]. Hence, splitting a complex model into several smaller ones of lower complexity facilitates readability of the complex model. If each submodel describes specific processes or distinctive parts of a complex process, also understanding the complete system is simpler. In a hierarchically structured control application, the higher-level components coordinate the execution of their subcomponents. Typical tasks are starting, stopping, and monitoring subcomponents, as well as handling errors. Hardware access is implemented at the lowest layer [7]. The architecture is illustrated in Fig. 1.

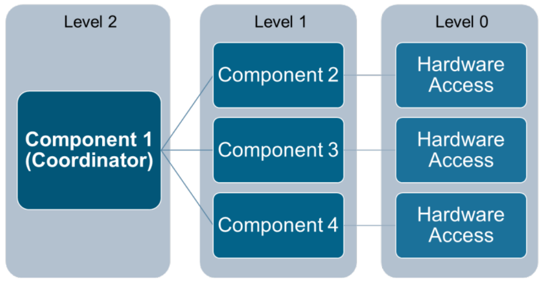


Fig. 1: The higher-level component 1 orchestrates components 2-4.

Hierarchically structured models have advantages in maintainability, as parts can be more easily replaced [8]. The decomposition forms the basis for an architecture that reaches from a general description at the highest abstraction layer to a detailed description at the lowest layer. The resulting models are better readable and comprehensible because the individual layers are less complex. Many control systems rely on a centralized coordinator and can be well reflected in a hierarchical structure [7]. Language features for hierarchical modeling are required to implement a layered design. We therefore outline the features of GRAFCET and IEC 61499 in the next sections. While GRAFCET (Section IV) relies on modelling multiple partial Grafcets and directly supports their coordination, IEC 61499 (Section V) ensures that all functionality is encapsulated in FB units that are coordinated via event and data flow.

IV. Structuring mechanisms of GRAFCET

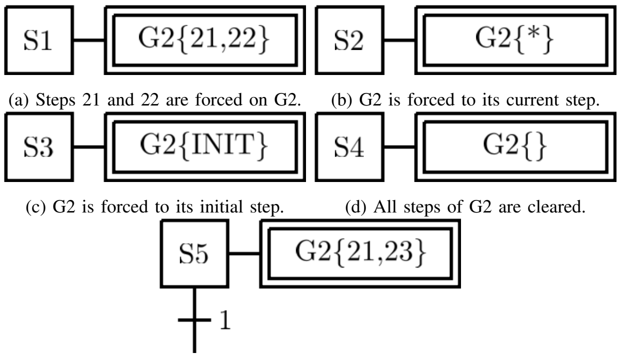
Grafcets can be structured via three distinct features:

1) *Forcing orders* are used to orchestrate partial Grafcets. The Grafcet that contains the forcing order is considered as the higher-level one (G1). When a forcing order is activated in a step of G1, this order forces a certain situation onto G2, the respective lower-level partial Grafcet. G2 cannot evolve freely, but is frozen and stays in the forced situation as long as the step containing the forcing order in G1 is active. It is not allowed that G2 in turn also forces a situation on G1, neither directly nor via a loop: While G1 forces G2, and G2 forces G3, G3 must not force G1 nor G2. Forcing orders have the highest priority when evaluating the functionality of a Grafcet. There is however no priority between various forcing orders, so it has to be ensured that a certain partial Grafcet is never forced by two different sources.
2) *Enclosing steps* contain a partial Grafcet. As soon as an enclosing step is entered, the initial steps of the contained partial Grafcet are activated. The enclosed partial Grafcet can evolve freely as long as the enclosing step is active. Once the enclosing step is deactivated, the enclosed partial Grafcet is cleared, i.e., all contained steps are also deactivated.
3) *Macro-steps* hide complexity because a sequence of steps is represented as a single step. They do not offer any additional functionality. They are used to split the functionality of a Grafcet to several IEC 61499 Basic FBs, when implementing Grafcets according to [6].

Based on examples, we will now further discuss the implementation of forcing orders and enclosing steps.

A. Forcing orders

Graphically, forcing is illustrated as a double-framed rectangle that is associated with a step. Forcing must be distinguished from the similar-looking actions, which were described in [6]. Forcing orders are noted as G#{...}, where G# is the name of the forced Grafcet and the forced situation is described in the brackets. As multiple steps of a Grafcet may be active simultaneously, it is possible to force several active steps as shown in Fig. 2a. G2 can also be forced to remain in its current state, the forcing then only prevents evolution of G2 (Fig. 2b). As illustrated in Fig. 2c, G2 may also be forced to its initial state. Finally, it is possible to



(a) Steps 21 and 22 are forced on G2. (b) G2 is forced to its current step. (c) G2 is forced to its initial step. (d) All steps of G2 are cleared. (e) Forcing order placed on an unstable step.

Fig. 2: Examples for steps with the supported forcing orders.

force an empty situation on G2 where all steps are deactivated (Fig. 2d). This is equivalent to switching off certain features or shutting down a process. Typical applications of forcing steps are initialization and switching the operation mode of a machine. As an active forcing order blocks the evolution of the subordinate Grafcet, the forcing order is often placed on an unstable step. Forcing has the highest priority when the Grafcet is evaluated. Therefore, the design pattern shown in Fig. 2e is used. G2 is forced and then the step is left immediately so that the situation of G2 can evolve freely. In S5, the Grafcet G2 is forced to a situation where step 21 and step 23 are active. As the transition condition is 1, S5 is left immediately and the forcing ends. G2 now has two active steps (21, 23) and can evolve.

B. Enclosing steps

Enclosing steps are represented graphically as a normal step with black corners as shown in the left image of Fig. 3. This step controls the activation and deactivation of one or more enclosed partial Grafcets with the same label, activating the steps of the enclosed Grafcet marked by an asterisk. The enclosed partial Grafcet can evolve freely as long as the enclosing step is active. As soon as the enclosing step is deactivated, also all steps of the enclosed Grafcet are deactivated. An enclosing step may include a Grafcet with other enclosing steps: If both enclosing steps have been active and the hierarchically higher enclosing step is deactivated, also both enclosed partial Grafcets will be deactivated. Although enclosing steps have a different graphical representation than normal steps and special functionality, all other properties of GRAFCET steps hold. Enclosing steps follow the notation X∗ of common steps and enclose one or more partial Grafcets that are referred to as X∗/G#. In Fig. 3, the enclosing step X9 contains the partial Grafcet X9/G2, which activates the initial step X9/X91 when X9 is activated.

An enclosing step can be the initial step of a Grafcet. It is then represented as a square with black corners and a double border (left image of Fig. 4). Upon first activation, such an *initial enclosing step* will activate the initial steps of its enclosed partial Grafcets (illustrated with a double border). For further activations, this step behaves like any enclosing step, activating all steps of the enclosed Grafcet that are marked with an asterisk. The initial step can be among these steps. An example is shown in Fig. 4.

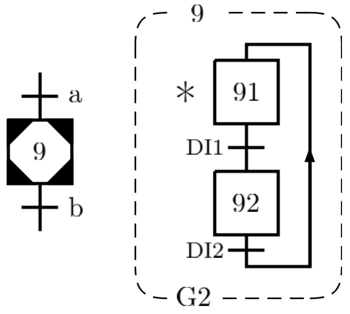


Fig. 3: Enclosing step X9 and its enclosed partial Grafcet X9/G2. The asterisk marks the initial step that is activated when entering X9.

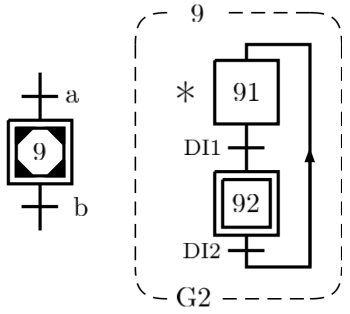


Fig. 4: Initial enclosing step and its enclosed partial Grafcet. When X9 is activated the first time, the initial step X9/X92 is activated. When X9 is activated again later, the step X9/91 will be activated (marked by an asterisk).

Only the translation of forcing orders will be covered in the following sections. Enclosing steps are transformed to forcing orders before the implementation. The behavior of an enclosing can be equivalently expressed in terms of forcing orders and transitions between steps (see Fig. 5). The forcing of the partial Grafcet G2 is designed to activate all steps that would be activated upon entering the respective enclosed step (i.e., all steps marked with *). Please note that this asterisk had a different meaning for forcing orders (Fig. 2), where * denotes the current step. The forcing is placed on an unstable step to allow the partial Grafcet to evolve freely. Finally, when the condition b for leaving the enclosing step is met, the enclosed Grafcet is forced to the empty situation. A variation of this pattern for an initial enclosing step is illustrated in Fig. 6,

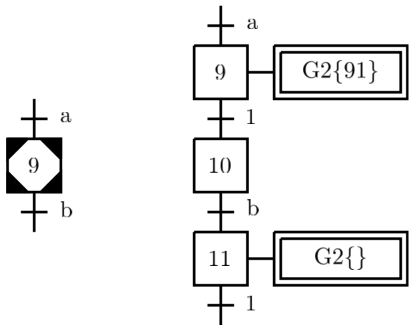


Fig. 5: Equivalence between the enclosing step (Fig. 3) and a Grafcet with forcing steps.

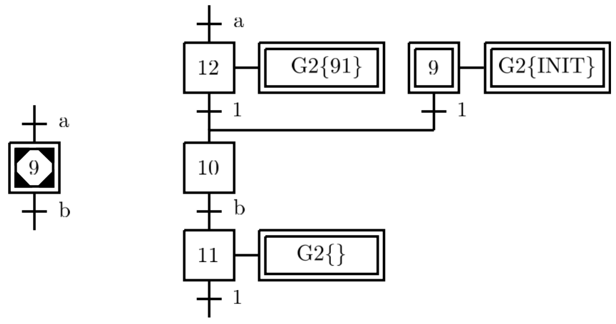


Fig. 6: Equivalence between initial enclosing step (Fig. 4) and forcing step. Forced Grafcet is X9/G2 and the starred step (marked with an asterisk) is X9/X91.

where the partial Grafcet is forced to the initial situation in an additional initial step X9. Because this initial step is unstable, G2 will immediately enter the idle step X10 after the forcing. G2 waits until the transition for leaving the enclosing step (b) fires. The initial step X9 will not be reached again, instead, step X12 will be entered, which will force the step G2/X91, mimicking the expected behavior of an enclosing step.

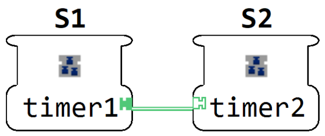
V. Structuring mechanisms of IEC 61499

In IEC 61499-models, interaction between application parts is only possible via interface pins with events and associated data connections. This ensures modular software units and provides an abstraction level between implementing an FB and using its instance. Typically, input event pins include initialization and execution requests, output events trigger subsequent application parts or confirm a successful operation.

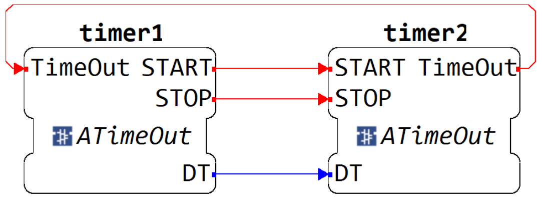
IEC 61499 does not comprise an explicit mechanism that is equivalent to forcing orders. Instead, developers manually control the execution of their application via events. In a layered application structure [11], a main coordinating Function Block controls the execution of other application parts. Basic FBs are used because their behavior can be controlled via the state-based Execution Control Chart (ECC). The evaluation of the ECC is triggered by an incoming event and, subsequently, other FBs are triggered by output events sent from this FB.

Using *adapters* is recommended for structured designs that clearly show the relation between hierarchical levels. Adapters group connections and are a method to structure the interface of blocks. They are bidirectional links and can be comprised of event and data connections. Adapter types are created as a special kind of FB type that always has two mirrored versions: a plug and a socket. Adapter connections are then created between the plug and the socket. The functionality of an adapter is illustrated in Fig. 7 with the adapter type `ATimeOut`, which can be used for connecting a timer block. The plug `timer1` is connected to the socket `timer2`. An element of this adapter is for instance denoted as `timer1.START` (for an event of the plug) or `timer2.DT` (for a data pin of the socket).

To structure control applications logically, FBs can be combined into hierarchical blocks, i.e., *subapplications (SubApps) or Composite FBs*. These blocks contain a network of FBs and are themselves blocks with an interface. Hierarchical blocks are stored in the library and are thus reusable.



(a) Adapter connection (green) of type ATimeOut from plug (timer1) to socket (timer2).



(b) Illustrating the adapter communication between timer1 and timer2. If block S1 sends an event START via the adapter, an output event is generated in the plug (timer1.START). The socket receives an input event (timer2.START) and forwards it to the block S2.

Fig. 7: An adapter structures the interface of a Function Block.

VI. Translation Patterns

IEC 61499 does not have any language features that are equivalent to forcing orders or enclosing steps, the structuring mechanisms of GRAFCET. Their behavior can though be mimicked with the translation patterns proposed in this section. As explained in Section IV, enclosing steps can be expressed using forcing orders, hence, only a pattern for forcing orders will be required. Please refer to [6] for the principles and translation patterns for Grafcets without structuring mechanisms.

A. Preparing the Grafcet model for translation

A key consideration for translation is the marking of a Grafcet. An active step in a Grafcet is denoted as marked. Grafcets can have several active steps simultaneously. GRAFCET steps are translated to states in IEC 61499's Execution Control Chart (ECC). As the ECC does not support multiple active states, multiple-marked Grafcets have to be split into several partial Grafcets with single marking. Forcing orders can cause multiple-marked partial Grafcets (e.g., Fig. 2a: forcing two steps simultaneously), which must be identified before proceeding to the implementation because only single-marked Grafcets can be translated. Multiple-marked situations must be divided into different partial Grafcets according to the rules in [6]. Within an FB, only a single-marked Grafcet with a sequence of steps can be implemented.

B. Implementing forcing orders of a higher-level Grafcet

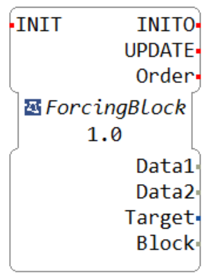
The translation pattern for forcing orders consists of two main parts, the forced partial Grafcet and the forcing order. In the IEC 61499-model, their interaction is modeled as events and data. The following two patterns translate a forcing command that is generated in the FB which implements the higher-level Grafcet. One or more other FBs will respond to it and implement the forced partial Grafcet.

Our forcing command is comprised of an event and associated data. Events trigger the execution of an FB and are essential for activating and deactivating application parts. To execute forcing orders, a block must be notified that it has to be executed (event trigger for Forcing), and which step shall be activated.

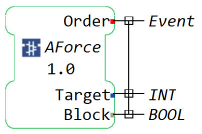
We suggest using the following signals:

- An event pin `Order` is used to send a forcing order to another FB. The event triggers the execution of this FB.
- An integer input `Target` defines the forcing target depending on the forcing types discussed in Section IV. The forced state is indicated by the step number. Each FB can only have one forced step number, as only a single ECC state can be active at a time. As a convention, we suggest to assign the step number of the original partial Grafcet. The integers 0 and -1 are used as reserved numbers and indicate a specific behavior: Target is set to -1 to force an empty situation and deactivate all steps (Fig. 2d). Target is set to 0 to indicate that the ECC remains in the current active state (Fig. 2b). This corresponds to blocking the evolution of a forced Grafcet without forcing a new situation.
- If the boolean input `Block` is TRUE, the evolution of the forced Grafcet has to be blocked. In the IEC 61499-implementation, transitions between ECC states that represent Grafcet steps are then not allowed.

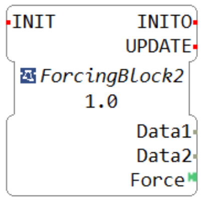
The interface of a forceable Basic FB is shown in Fig. 8a. The pins that are required for forcing are added individually: Order, Target, and Block. We however suggest using an adapter type for these connections. A dedicated forcing adapter (Fig. 8b) ensures that only a single typed connection is needed, thus reducing error-proneness of the design. Adapters are also easily recognizable in the application model and show a clear relation between the hierarchical application parts. When using the adapter in the interface of our Basic FB (Fig. 8c), only one plug pin needs to be added. The same adapter connection is usable for all types of forcing.



(a) FB Interface with individual pins.



(b) Adapter type AForce for implementing forcing orders. It groups the three required pins.

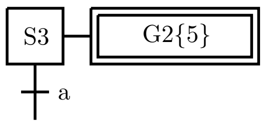


(c) FB Interface with Adapter of type AForce.

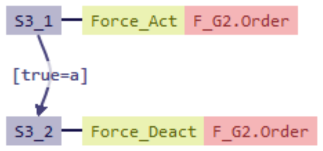
Fig. 8: Basic FB with an interface for forcing. An event Order triggers a forcing order. This order is detailed via the data pins Target and Block.

1) Evolution-blocking Forcing Order: Consider the example forcing order presented in Fig. 9a. The presented forcing order forces the Grafcet G2 to activate step 5 and prevents the evolution of this forced Grafcet until the condition `a` holds, i.e., the outgoing transition from S3 is crossed. In IEC 61499, we use an adapter connection of our type `AForce`. The adapter pin is named `F_G2`, hence, the output event pin Force is referred to as `F_G2.Force`. As IEC 61499 does not specify a mechanism for freezing an ECC, the boolean variable `F_G2.Block` is used to block the evolution of the target partial Grafcet. Implementing the forcing behavior requires two separate ECC states as shown in Fig. 9b. The states have an action to activate and deactivate the forcing, in which the respective algorithm sets the data accordingly. Both actions send an event `Order`.

The forcing order parameters are set up in the algorithm `Force_Act`. To activate the forcing order, the evolution is blocked (`F_G2.Block := 1`) and the forced step is set to 5 (`F_G2.Target := 5`). In the second algorithm `Force_Deact`, the forcing order is deactivated: The target step is set to the forced Grafcet's current step (reserved number, `F_G2.Target := 0`) and evolution is allowed (`F_G2.Block := 0`). With this ECC pattern, the forcing



(a) Forcing order.

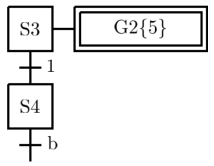


(b) ECC Implementation with two states.

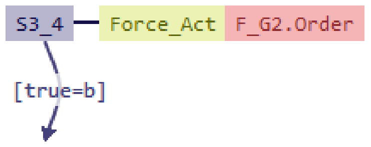
Fig. 9: Translation pattern for issuing a forcing order. The algorithms block the evolution of the subordinate Grafcet G2.

orders presented in Fig. 2a-2c are covered by typing the correct target state in the activation algorithm `Force_Act`. For Fig. 2d, the target state has to be set to -1. All of these forcing orders were placed on a stable step.

2) Non-Blocking Forcing Order: A special situation occurs when the forcing order is placed on an unstable step. The expected behavior is to apply the forcing order and to then leave the step. Such a forcing order is used for instance to model the behavior of enclosing steps with forcing orders (see Fig. 5-6. The corresponding pattern is presented in Fig. 10, where the forcing order (Fig. 10a) is translated to a state with a single action (Fig. 10b). The algorithm `Force_Act` assigns the corresponding target step and sets the blocking bit to 0 (`Force.Block := 0`) to allow evolution. A single output event Order is sufficient.



(a) Forcing order.



(b) ECC Implementation.

Fig. 10: Translation pattern for a forcing order placed on an unstable step.

C. Implementing the forced partial Grafcet

For better abstraction and maintainability, the implementation of the forced partial Grafcet should be agnostic of the type of forcing that is applied. This abstraction is also present in the Grafcet model, as a forced partial Grafcet does not model any details about the forcing. A forced block requires an adapter socket of type AForce at the interface, which we named `F`.

The behavior of the forced Grafcet is implemented in the ECC. The pattern is illustrated in Fig. 11, which shows the ECC for the partial Grafcet G2 from Fig. 3. Each Basic FB expects an initialization event INIT to enter the state Init. Necessary algorithms, which for example initialize variables, can be executed in this state's action. Due to the outgoing 1-transition, the state Init is unstable and Initialized is set active after the algorithm was executed. The FB is now waiting for the next event. Compared to an FB without forcing (as in [6]), the ECC has an additional forcing section that is comprised of the states Forcing_hub and Emptying. These states handle the activation and deactivation of a partial Grafcet. An ECC following this pattern can handle all types of forcing. We will now discuss the execution behavior based on examples.

1) Evolution-blocking Forcing Order: The data input representing the blocking boolean, F.Block, is true. As soon as the FB receives an Order event, the state Forcing_hub is entered. The algorithm associated with this state resets the values of all outputs. The target state is read from the respective data input (Target). No event conditions are modeled at the outgoing transitions from Forcing_hub, so the target state is entered immediately. In the associated algorithm, required variables will be set and the outputs will be updated with the event UPDATE. Leaving the forced state was prevented by adding an additional condition (AND not (F.Block)) to each transition between states that implement Grafcet steps. From all these states, an incoming and outgoing transition with Forcing_hub is implemented. In our example, these are the states E91 and E92. An active state is left as soon as the FB receives a forcing order with a different target state. From Forcing_hub, the state targeted in the forcing order is entered.

The forcing order ends when the higher-level Grafcet leaves the step that issued a forcing order. At that time, another forcing command is received that deactivates the ECC (F.Target is -1). All outputs will be reset at Forcing_hub and the output data are updated in Emptying by sending an output event. The FB remains idling in the state Initialized.

2) Non-blocking Forcing Order: When receiving a forcing order that was placed on an unstable step, F.Block is false. Evolution is then possible because the transition conditions between E91 and E92 can evaluate to true.

VII. Evaluation Example

We will now illustrate how to implement the presented structuring mechanisms based on the example presented in Annex B4 of the GRAFCET standard [1], which is an automated weighing-mixing system. The system operation is described in the standard. A similar Grafcet, yet without structuring mechanisms, was implemented in IEC 61499 in [6].

A. Hierarchical Grafcet of the weighing-mixing system

The Grafcet with an enclosing step is presented in Fig. 12. This system is structured according to several operation modes: Safety stop mode, Manual mode and Automatic mode, represented by steps D1, A6, and F1, respectively. The system starts in safety stop mode (D1). Manual controls can be enabled (action EMC) in Manual Mode (A6). This mode is activated via the manual mode selector switch SSManu unless the emergency stop push-button PBES is pushed. If the selector switch for automatic mode SSAuto is now activated and several safety conditions (in terms of the variables of the process) are met, the Automatic mode is activated (partial Grafcet F1). The automatic mode starts in stop position (F1/X0) where the system waits for a signal from the push-button CS. This position is also reached after every operation cycle. As soon as the button is pressed, the normal operation begins, which is presented in the partial Grafcets Weighing, Transport, and Mixing. A cycle ends when the partial Grafcet Mixing reaches step X44 (leaving the inner enclosed step). Please note that the enclosed Grafcets are also deactivated when the hierarchically higher enclosing step is left. Therefore, the automatic mode is interrupted as soon as manual control is enabled via the selector switch SSManu. Finally, the emergency stop push-button ends any current activity instantly and the system returns to the safety stop step.

B. Implementing the control software in IEC 61499

First, the two enclosing steps are modelled as forcing steps according to Fig. 5. Each partial Grafcet consists of a sequence of steps without multiple-marking and is therefore implemented within one Basic FB. We now exemplarily show the implementation of the Grafcets Weighing and GM. Their interfaces (in Fig. 13) contain an AForce socket because both Grafcets are forced by a hierarchically higher Grafcet. GM in turn forces three other Grafcets, hence, three AForce plugs are added. Each FB has additional data pins for transition conditions (inputs) and actions (outputs). No global variables are available, so also the active state of an ECC has to be transmitted via data pins (X23 and X44). Finally, events for initialization and operation are added. Only the Weighing FB has an UPDATE event because its steps have associated actions that require refreshing output data. The adapters used to coordinate the Weighing, Transport, and Mixing have been named FW, FT, and FM, respectively. Therefore, the forcing orders are sent via FX.Order (X designating W, T, or M).

The behavior of our FBs is implemented in their ECCs (Fig. 14) which resemble the template presented in Fig. 11. The function was described in Section VI. After the initialization phase, they idle until a forcing order is received. The target state is entered via Forcing_hub. In our case, the forcing orders originate from an enclosing step and do not block the evolution (F.Block is 0). The ECC of GM shows the implementation of an enclosing step which, as shown in Fig. 5, is modeled by two forcing orders on an unstable step, which have been translated to states E2 and E2_des using the translation pattern in Fig. 10. Please note that the steps '0' and '1' were renamed to E1 and E2 because, in our implementation, the Target '0' is a reserved number representing that a forced Grafcet must remain in its current

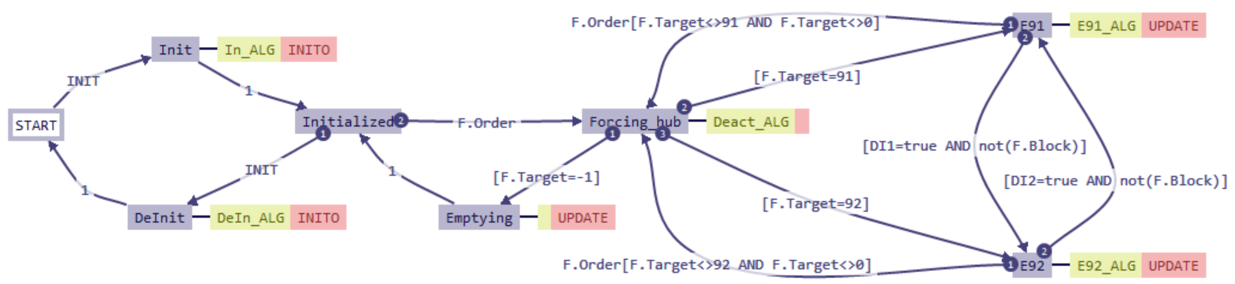


Fig. 11: ECC implementing the forceable partial Grafcet G2 from Fig. 3. Its structure is based on [6].

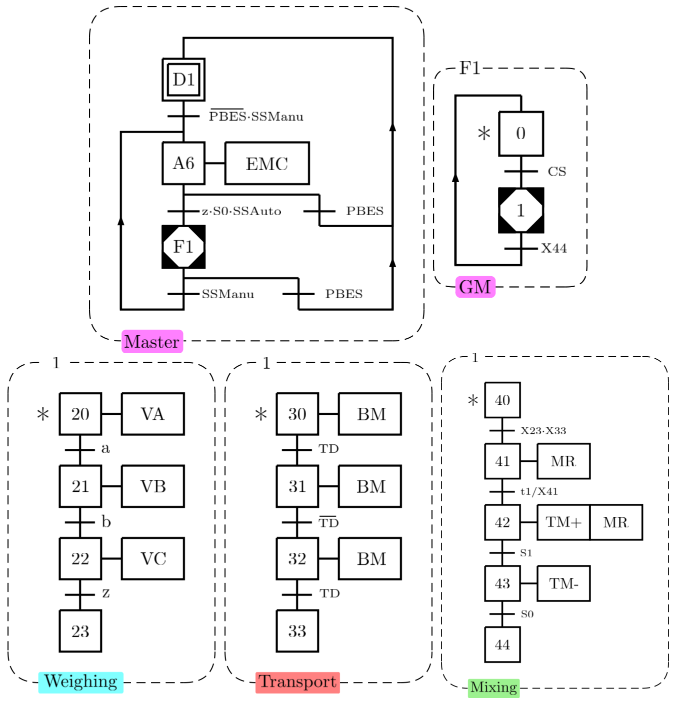
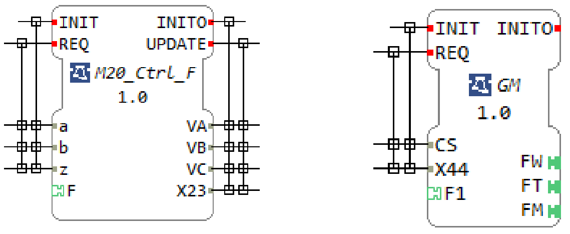


Fig. 12: Grafcet of the system under study with three hierarchical levels via nested enclosures.



(a) Weighing FB.

(b) GM FB.

Fig. 13: FB Interfaces for Weighing and GM Grafcets.

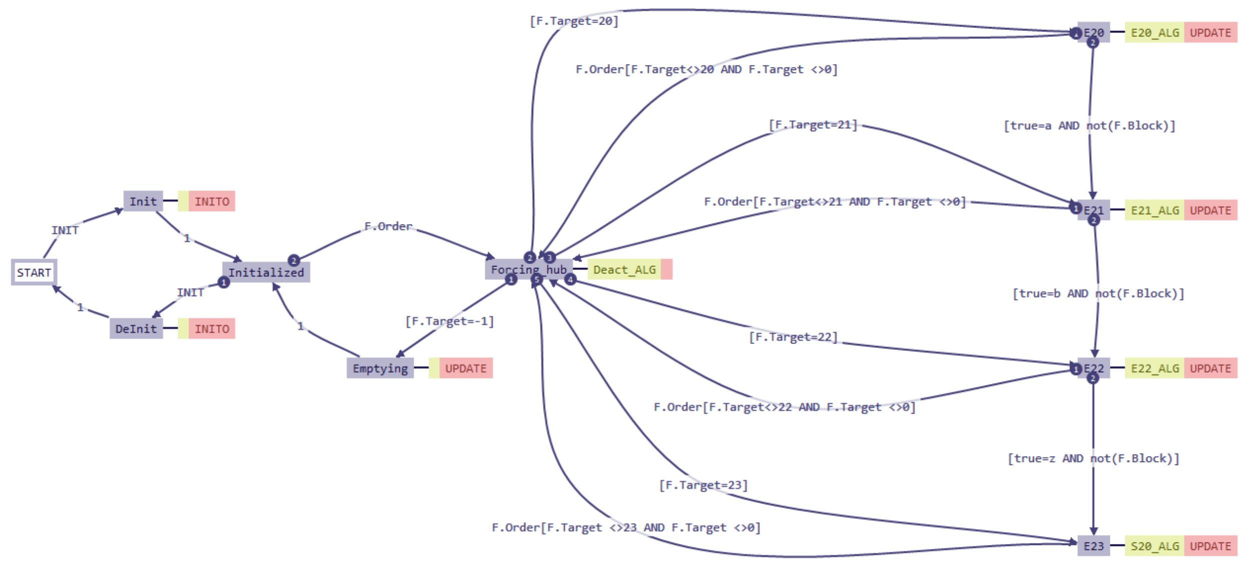
step. The forcing instructions are set and reset in separate algorithms to allow reusing them in the `Forcing_hub` state when a forcing order is received.

The IEC 61499-application for the weighing-mixing system (Fig. 15) is composed of four SubApps: Master, which contains the FB implementing the functioning of Grafcets Master and GM, and Weighing, Transport and Mixing which implement the functioning of the respective Grafcet. Within a SubApp, sensor data is read, our Basic FB controls the execution, and output values are written to actuators. As an example, Fig. 16 shows the Weighing SubApp. The remaining parts of the implementation are equivalent and can be obtained by applying the principles presented in this paper. Additionally, the results presented in [6] are helpful for fully implementing a distributed Grafcet in IEC 61499.

VIII. Conclusion and Outlook

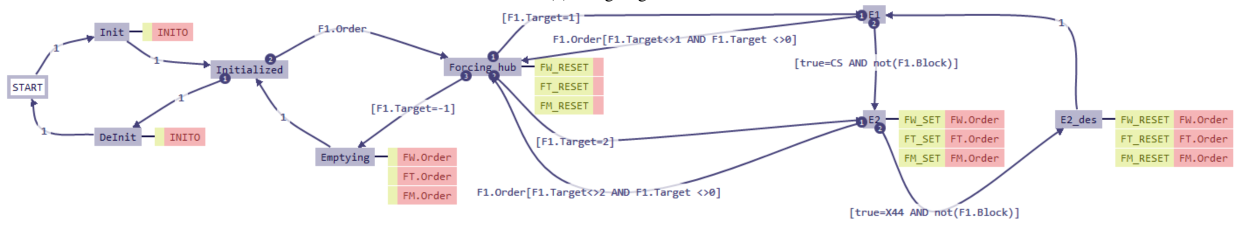
The systematic implementation of GRAFCET's native structuring mechanisms in the IEC 61499 standard enables engineers to implement distributed control software based on GRAFCET models with hierarchical elements. A direct translation between the two language is not feasible because the languages have a distinct language scope. Forcing orders and enclosing steps have to be expressed via adapters and SubApps, the native structuring elements of IEC 61499. We showed our systematic approach based on an application example. This paper complements our previous work with translation patterns for Grafcets. Combined, they constitute a complete set of translating patterns for fully implementing any Grafcet model in a IEC 61499 application. The translation patterns can be applied systematically and the guidelines are useful for domain experts. A limitation of our approach is the propagation of changes from the GRAFCET to the implementation. Maintenance of complex projects may only be possible in the IEC 61499 software. An automated translation may be thus required. Furthermore, the forcing may increase the complexity of Basic FBs compared to a design without hierarchical structures. However, our approach takes currently available design patterns into account. Finally, our implementation does not consider error-handling of the IEC 61499 system, e.g., regarding the communication between devices. If required, it can be added to the implementation.

Some aspects still require further investigation. The event generation policy rules the execution of the algorithms and may affect the application performance, but has not been discussed in our implementation. Furthermore, the effects of the distribution on the performance have not been evaluated. Studies that include the effect of the communication delays in conjunction with the event generation policy are interesting to develop.



[F.Target=20]
F.Order[F.Target<>20 AND F.Target <>0]
[F.Target=21]
F.Order[F.Target<>21 AND F.Target <>0]
[true=a AND not(F.Block)]
[true=b AND not(F.Block)]
[true=z AND not(F.Block)]
[F.Target=22]
F.Order[F.Target<>22 AND F.Target <>0]
[F.Target=23]
F.Order[F.Target <>23 AND F.Target <>0]
[F.Target=-1]
F.Order
INIT
START
Init
INITO
Initialized
DeInit
Forcing_hub
Deact_ALG
Emptying
UPDATE
E20
E20_ALG
E21
E21_ALG
E22
E22_ALG
E23
S20_ALG

(a) Weighing ECC.



(b) GM ECC.

Fig. 14: ECC diagrams for implementing the Grafcets Weighing and GM.

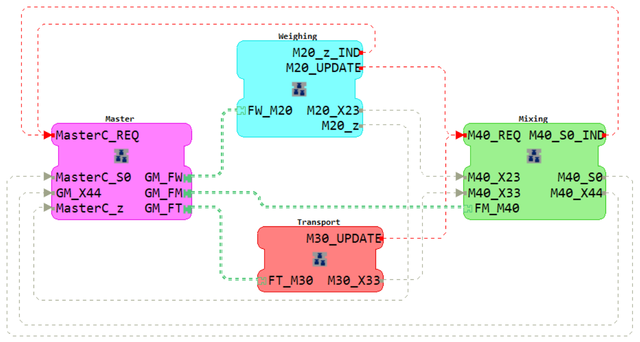


Fig. 15: IEC 61499 application of the Grafcet presented in Fig. 12.

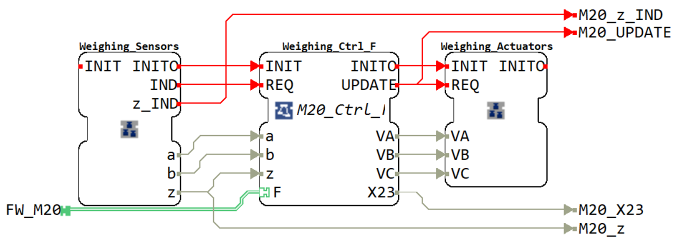


Fig. 16: Subapplication of the weighing unit.

References

[1] IEC, “IEC 60848: GRAFCET specification language for sequential function charts,” 2002.

[2] R. Julius, M. Schürenberg, F. Schumacher, and A. Fay, “Transformation of GRAFCET to PLC code including hierarchical structures,” *Control Engineering Practice*, vol. 64, pp. 173–194, 2017.

[3] ADEPA, “Gemma (Guide d’Etude des Modes de Marches et d’Arrêts),” 1981.

[4] N. Hagge and B. Wagner, “Implementation alternatives for the omac state machines using iec 61499,” in *2008 IEEE International Conference on Emerging Technologies and Factory Automation*, pp. 215–220, 2008.

[5] IEC TC65/WG6, “Iec 61499-1, function blocks - part 1: architecture v2.0: Edition 2.0.”

[6] O. Miguel-Escrig, J.-A. Romero-Pérez, B. Wiesmayr, and A. Zoitl, “Distributed implementation of grafcets through iec 61499,” in *2020 25th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, pp. 402–409, IEEE, 2020.

[7] A. Zoitl and R. W. Lewis, *Modelling control systems using IEC 61499*, vol. 95 of *IET Control engineering series*. London: IET, 2. ed. ed., 2014.

[8] B. Brandenbourger and F. Durand, “Design pattern for decomposition or aggregation of automation systems into hierarchy levels,” in *2018 IEEE 23rd Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, pp. 895–901, IEEE, 2018.

[9] F. Schumacher and A. Fay, “Formal representation of GRAFCET to automatically generate control code,” *Control Engineering Practice*, vol. 33, pp. 84–93, 2014.

[10] IEC, “IEC 61131 - programmable controllers, part 3: Programming languages,” 2013.

[11] A. Zoitl and H. Prähofer, “Guidelines and patterns for building hierarchical automation solutions in the IEC 61499 modeling language,” *IEEE Trans. on Ind. Inf.*, vol. 9, no. 4, pp. 2387–2396, 2013.

[12] B. Wiesmayr, L. Sonnleithner, and A. Zoitl, “Structuring distributed control applications for adaptability,” in *Proceedings of the 3rd IEEE Int. Conf. on Industrial Cyber Physical Systems (ICPS 2020)*, 2020.

[13] T. Green and M. Petre, “Usability analysis of visual programming environments: A ‘cognitive dimensions’ framework,” *Journal of Visual Languages & Computing*, vol. 7, no. 2, pp. 131–174, 1996.

[14] P. Forbrig and C. Märtin, “Elaboration on terms and techniques for reuse of submodels for task and workflow specifications,” in *Human-Computer Interaction. Theory, Design, Development and Practice* (M. Kurosu, ed.), (Cham), pp. 467–475, Springer International Publishing, 2016.