Distributed implementation of Grafcets through IEC 61499

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

Bianca Wiesmayr, Alois Zoitl, *Member, IEEE*
LIT Cyber-Physical Systems Lab
Johannes Kepler University Linz
Linz, Austria
{bianca.wiesmayr, alois.zoitl}@jku.at

***Abstract*—A Grafcet is a standardized model for describing the behavior of systems which is popular among automation engineers. As the Grafcet standard excludes implementation details, the models are typically translated to automation software. Such software was traditionally programmed in one of the languages specified in IEC 61131-3. Nowadays, automation software is increasingly modelled in IEC 61499 which facilitates designing distributed control systems. In this paper, we define a standardized translation methodology, so that automation engineers can benefit from the advantages of IEC 61499 while continuing to use Grafcet. We discuss the differences between Grafcet and IEC 61499. We translated a Grafcet model into an IEC 61499 application to illustrate the process and derive guidelines for application designers. For the core concepts of Grafcet, we present the corresponding structure in IEC 61499.**

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

I. Introduction

Models are used widely in scientific fields because they are designed to abstract from highly complex real-world phenomena. In engineering, modelling approaches are increasingly required to design large and complex systems [1]. In industry, several levels of modelling can be attained depending on the goal that has to be achieved. In manufacturing industries, SysML [2] has emerged as one of the most complete tools for describing the whole productive chain. On the other hand, model-driven engineering in automation is based on the use of different modelling approaches, some alternatives are Petri nets, statecharts or GRAFCET, each of them with its own description capabilities. Among them, GRAFCET is one of the most extensive and provides a very intuitive and descriptive graphical visualization. This standard [3] was designed as a graphical modelling language that is tailored for a specific domain (domain-specific modelling language, DSML). It is used to formally specify the functioning of a system in so-called "Grafcets", which are state-based models. They are comparable to statecharts [4], but are based on Petri nets and better suitable for the domain because they are available as an industrial standard (IEC 60848) [5].

Misinterpretations of the system functionality may be avoided when using well-defined modelling languages instead of textual requirements. In general, DSMLs may improve productivity and quality of the software and facilitate platform-independent design. Formal specifications such as state machines provide a basis for automated validation and verification of models, which can reveal errors at early stages of the development process [6]. These advantages of DSMLs explain the popularity of GRAFCET among automation engineers.

Even though GRAFCET could be used for modelling distributed control systems, traditionally the controllers modelled with Grafcet have been implemented in a centralized way, i.e., the control algorithm resulting from the model is executed in a single control device. This restriction is not an intrinsic feature of the GRAFCET standard whose description level defines the system behavior but not the implementation details. This limitations in the implementations are introduced by the software resources available for implementing the control algorithms from this kind of models.

In industries involving automation processes, control algorithms modelled with Grafcet are usually implemented following the standard IEC 61131 [7], which still constitutes the main driving force in these environments [8]. As the standard supports a variety of graphical and textual programming languages, developers can flexibly choose the most suitable language for each part of a software project. Generally, the available development environments cover functionality for design, validation, and implementation aspects. The recently published object-oriented extensions [9] further increase the scope of the standard. Thanks to this versatility, the know-how in industry is nowadays consolidated and widespread. Limitations of these programming languages include the difficulty of encapsulation and the lack of support for distributed systems.

New challenging tasks for software development are emerging due to the increased adoption of Industry-4.0-principles that require designing distributed control applications. Such software runs on numerous hardware nodes which form a System of Systems. In such environments, the newer standard IEC 61499 may be more suitable, as it was designed as an executable DSML for networked control systems. The application design is independent of a system configuration, which allows designing distributed control systems. The centralized application model can be distributed across multiple devices, so that each part of the application can run on a different resource.

The aforementioned features of IEC 61499 open the door for the distributed implementation of Grafcet models. This paper proposes a well-defined methodology to create control software based on IEC 61499 which behaves like the GRAFCET specification of the system. It allows automation engineers to create distributed designs, while providing application developers access to a standardized, high-level modelling technique. The model behavior of both is compared in Section III to provide a detailed understanding of the underlying concepts. Section IV contains translation patterns for implementing language features of GRAFCET in IEC 61499. An application example (Section V) is implemented in Eclipse 4diac [10], a development environment for IEC 61499-based applications to show the applicability of the developed translation patterns. The paper is concluded in Section VI.

II. Related Work

When selecting a target language for implementing Grafcets, readability and maintainability are important factors [5]. Grafcets have therefore been mostly translated to IEC 61131 programs using the languages SFC (graphical) and ST (Structured Text, textual).

An approach that allows translating a Grafcet into ST code is discussed in [11] where the elements and the behavior of a Grafcet are described by a Control Interpreted Petri Net (CIPN). Thus, an algorithm that assures the correct behavior of CIPNs, which can be easily translated to ST, can be also applied to Grafcets. Similarly, some publications treat the implementation of Signal Interpreted Petri Nets (SIPN) in SFC. SIPN partially describe the functionality of GRAFCET [12], [13].

An approach for direct translation from GRAFCET to SFC was shown for example in [14]. A normalization was applied to the Grafcet to remove hierarchical elements because, unlike GRAFCET, SFC does not support hierarchical elements. Hierarchy is however a common tool to reduce the system complexity by abstracting from lower-level details and to facilitate understanding models of large systems. IEC 60848 has several mechanisms for creating hierarchical models by using language elements such as macrosteps that enclose other Grafcets and so-called partial Grafcets [3]. A translation algorithm to ST that preserves the (hierarchial) structure of the Grafcet was therefore proposed [5]. Other languages were treated in [15], [16], but they have poorer readability.

Despite the similarities between GRAFCET and SFC, this programming language has some limitations, hence, no method for fully translating Grafcet models to SFC language while retaining hierarchical structures exists. Some approaches, as in [14], include the hierarchical elements within a single Grafcet model, which allows its translation to PLC code. [5].

IEC 61499 is a promising candidate for implementing Grafcet models [5]. The execution is event-based and therefore matches GRAFCET better than the cyclic model of IEC 61131, and hierarchical elements are supported. While no direct transformation from GRAFCET to IEC 61499 is hitherto available, IEC 61131-3 code could be reused within individual FBs. This approach however does not allow creating distributed control nor reusing FBs. Additionally, translations of IEC 61131-programs to IEC 61499-models have been performed for FB networks [17] and for SFC [18]. IEC 61499-models were compared to statecharts and Petri nets in [19], resulting in an approach to integrate Petri nets and IEC 61499 [20].

The goal of this paper is to provide structured guidelines for creating an IEC 61499-application directly from a GRAFCET specification to take a step towards automated translation. Language features are carefully analyzed in order to create modular, well-structured applications.

III. Comparing the Model Behavior

GRAFCET is a modelling language for describing the behavior of a given system. The main model elements are steps, which can have some associated actions. The steps are connected via directed links which have a transition with an associated condition. In IEC 61499-models, an application is defined completely via Function Blocks (FB) and their interconnections. Among the different types of FB, the Basic Function Block (BFB) has a state diagram, which is called "Execution Control Chart" (ECC) and is semantically similar to GRAFCET. An ECC is composed of several states, which can have some associated actions. States are connected via transitions with associated conditions.

GRAFCET is proposed as a tool for modelling IEC 61499 applications because their main characteristics are similar. The IEC 61499 ECC is based on IEC 61131 SFC, which was again derived from GRAFCET [21]. Nevertheless, as some differences exist between both languages, they will be explained based on the two example graphs shown in Fig. 1.

1) Number of active states in the model: When entering parallel sequences, the ECC evaluates the transitions sequentially starting from the highest priority. As a result, exactly one state is active at any time. In contrast, GRAFCET clears all transitions whose condition evaluates to TRUE and all destination states are marked as active ("multiple marking"). In our graphical example, a parallel sequence occurs after S1. Let $a = 2$ and $b = 3$, then the GRAFCET sets S2 and S3 active. The ECC only sets S2 active, because the transition from S1 to S2 has the highest priority (shown in the circle at the transition origin). The Grafcet is equivalent to the ECC if the condition $b \geq 1$ is replaced by $a < 1$ && $b \geq 1$, i.e., GRAFCET's selection of sequences is equivalent to ECC if conditions are exclusive.

2) Structuring mechanisms: Structuring complex systems hierarchically creates an abstract model of high-level behavior. IEC 61499 applications are composed of FB instances. Within these FBs, an ECC can control the behavior of individual blocks. Unlike other state-based models, the ECC itself does not support hierarchy.

GRAFCET modelling introduces some mechanisms that allow regulating the behavior and structure. These mechanisms are based on the fact that the whole behavior of a system can be defined in the form of different partial Grafcets with the proper interconnection and hierarchy between them. The

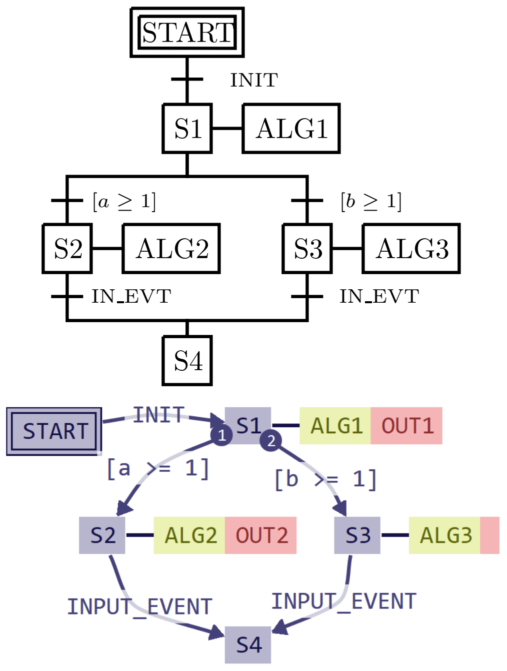

Fig. 1. Basic example for comparing the language specifications of GRAFCET (top) and ECC (bottom). The models are composed of the same language elements, but have a different execution behavior.

specific mechanisms to structure applications are forcing steps in other partial Grafcets or encapsulating the functioning of other partial Grafcets within a step. The encapsulation of functionality can be done either by an enclosure, which will execute a partial Grafcet as long as the enclosing step is active, or by a macro-step, which is a compact representation of a partial Grafcet that is executed from start to finish.

3) Action Elements: In both models, a state or step may have any number of actions. Each ECC action can consist of an algorithm and an output event, while GRAFCET actions deal with the modification of system variables or executing algorithms. As models in IEC 61499 are executed event-driven, FBs send output events to trigger the following FBs. In Fig. 1, this is reflected in the actions associated to S1 and S2. While they take the form of algorithms and output event in the ECC, they are modelled as an action in the Grafcet, which may modify a variable that works as a signal. This fact highlights a difference in the way that both standards treat data. While IEC 61499 generally uses events as signals, Grafcets (and their implementations in IEC 61131) usually handle signals by "parametrizing" their data, for instance, assigning a boolean value to create a rising/falling edge.

4) Transient steps: Upon entering an ECC state, the corresponding action is executed. Some types of actions in GRAFCET are only executed when the corresponding step is stable, i.e., all outgoing transitions evaluate to false. Hence, the step remains active until an associated condition of the enabled transitions becomes true. In our example, consider the START state and assume $a = 2$. As soon as an INIT event occurs, state S1 is reached and the left transition is crossed. S1 is transient (i.e., unstable) and its action is not executed, but S2 is stable. Exceptions are "Actions on activation". Like ECC actions, they are executed once when their execution condition is fulfilled.

Some GRAFCET principles do not have a corresponding item in IEC 61499-models or are postulated differently:

5) Nature of the actions: GRAFCET modelling provides a compact representation of actions, independently of where and how they will be executed. Nevertheless, IEC 61499 takes care of the nature of the action, requiring a different implementation. GRAFCET actions that involve resources or services to be deployed must be implemented with ECC actions that contain at least an output event. However, if a GRAFCET action does not require these services or only affects its behavior, it can be implemented in the algorithmic part of an ECC action.

6) Action types: GRAFCET actions can be modelled as stored or continuous actions. Stored actions are executed once when the activation condition is fulfilled (upon entering/leaving a step, receiving an event, or crossing a transition). In contrast, continuous actions are executed as long as the activation condition is fulfilled, which can be just being in a specific step or, once within a given step, fulfilling another boolean expression which can involve variables and/or temporal constraints. Nevertheless, within an ECC, only one type of actions is allowed, which are all executed when their corresponding state is entered.

7) Time-triggered conditions: GRAFCET allows defining time-dependant conditions for transitions and action executions. For example, a transition is cleared after the source step has been active for the specified time. In IEC 61499, blocking the execution of FBs is forbidden and therefore waiting is not possible.

IV. Translation Patterns

The differences between GRAFCET and ECC have to be considered when translating models. As shown in Fig. 1, an ECC can implement the functionality of several GRAFCET steps. Developers should avoid implementing the system behavior in a single FB/ECC, as this hinders reuse and maintainability of the design [22]. When creating an IEC 61499 implementation, not only requirements concerning application distribution and/or modularity need to be considered, but also language constraints. While IEC 61499-applications are designed to be distributable across several devices, this is not possible for individual FBs.

We therefore derived several patterns for common GRAFCET features that need to be translated to IEC 61499 and we outline guidelines for structuring the resulting implementation. The patterns refer to two categories: application structure patterns and functionality implementation patterns. Examples of these patterns are included in Figs. 2 to 12.

A. Application structure patterns

1) Distributing the Grafcet behavior across subapplications: Steps belonging to a modular unit and Grafcet parts

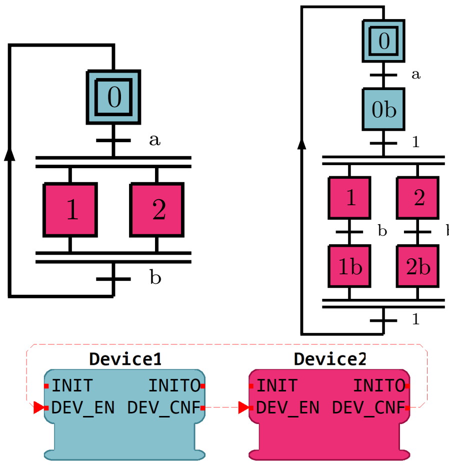

Fig. 2. Normalization of Grafcet model to be suitable for implementation in IEC 61499.

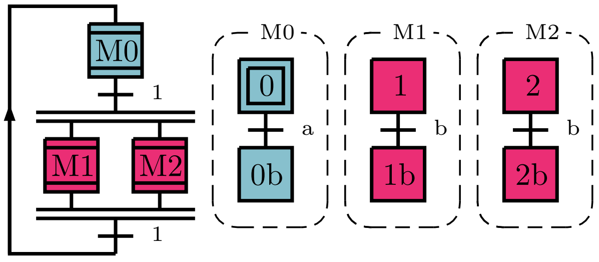

Fig. 3. Macrostep synthesis of normalized Grafcet model.

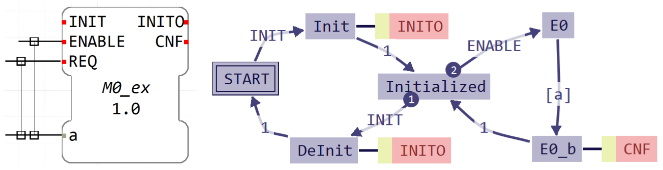

Fig. 4. Translating a normalized macrostep to a BFB.

that will run on separate devices should first be identified in the Grafcet model. To envisage a standardized implementation, this classification identifies the model parts that will later be implemented as individual subapplications to allow reuse and distribution. Subapplications contain a network of FBs and are blocks with an interface (like FBs). The interface of these subapplications will be comprised of (1) input event ports that allow initialization or execution requests, (2) output events that trigger subsequent application parts and confirm the completion of actions and (3) ingoing and outgoing data connections from/to other subapplications with parameters for correct functioning.

Inside each subapplication, we suggest to implement the Grafcet logic within the ECC of one or several basic FBs. They are part of the subapplication network, together with FBs dedicated to input data collection, output modification, and additional required functionalities (e.g., timers).

It is simpler to implement the effect of Grafcet transitions within an ECC. We therefore suggest modifying the original Grafcet so that distributed or modular elements are connected by a transition whose condition is always true. This is achieved by additional empty states at the end of each single marking distributed subapplication. That transition can thus be directly modelled by an event connection in the IEC 61499-application. An example of this modification is presented in Fig. 2, where a Grafcet consisting of two modules has been modified to include additional steps and an always true condition between different modular parts with single marking. The individual parts are highlighted by colors. Each part will be implemented within a different subapplication linked by an event connection.

2) Distributing functionality using Macrosteps: Macrosteps are a structuring mechanism in GRAFCET that is useful for compacting sequences of steps, defined by a partial GRAFCET, into a single step without adding any other special functionality. This structuring mechanism from Grafcet will be used to identify the number of BFBs needed in each subapplication. As ECC only supports one active state at a given time, the different processes within a subapplication must be divided into chains where only single marking is allowed. Continuing the example presented in Fig. 2, the compact representation will be composed of three macrosteps as shown in Fig. 3, where a macrostep has been assigned to each single-marked line.

Each of these macrosteps can now be implemented systematically as an ECC that is comprised of (1) initialization procedures, and (2) control algorithms. In Fig. 4, the macrostep M0 from the preceding example is presented. In the left part, a sequence of states initializes the FB until an idle state "Initialized" is reached. The control algorithms (right part) are executed as soon as an event arrives at the "ENABLE" input. The FB updates its variables from the input upon receiving an event at the "REQ" input, thus, the behavior of the algorithms may vary over time. The REQ event is sent as an output event by the preceding FBs in the network. This request event allows to implement different execution policies such as periodic calls to algorithms, cyclic calls, call-on-variable-updates, round robin in parallel processes, etc. by correctly handling the events. After the execution of the sequence has finished, an output event is sent at the "CNF" port right before returning to the idle state. This event will initiate the next macrostep and trigger the execution of another FB.

3) Parallelism and synchronizing sequences: By splitting the Grafcet behavior across different FBs, we have recreated the behavior of sequences of steps where only a single step is active. Nevertheless, GRAFCET supports parallelism and synchronization of sequences, behavior that can be mimicked by properly interconnecting the instantiated BFBs that implement the Grafcet logic.

In Fig. 5, both parallelism and synchronization are illustrated. Parallelizing the execution is supported by a fanout of output events. In our example, the source "CNF" event is connected to the "ENABLE" event of every BFB that should work in parallel. Synchronization of sequences is supported by including the block E_REND. This block sends an output event after each input port has received an event. Both, parallelism and synchronization, can be established between subapplications or between processes inside a subapplication, because the interfaces of subapplications and FBs are structurally equivalent.

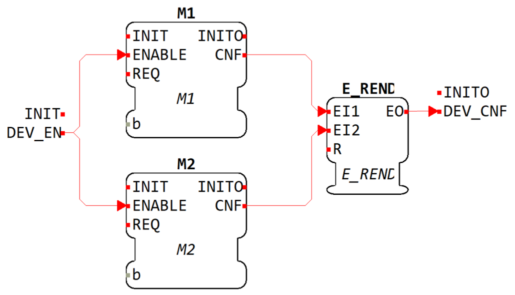

Fig. 5. Parallelism and synchronization in IEC 61499.

B. Functionality implementation patterns

1) Stored actions: To represent the various types of stored actions, additional states and proper transitions may be needed in the ECC. *Actions on activation* are easily translatable, because their behavior is equivalent to ECC actions. Both are defined to be activated upon state entry, even if the state is transient. *Actions on deactivation* can be modelled by introducing an additional state that holds the action. This state is entered when the exit conditions are met (see Fig. 6). *Actions on event* can be modelled by an additional state that holds the action. This state is entered whenever the trigger event is received and the ECC then returns to the original or idle state (see Fig. 7). Finally, *actions at the clearing* can be modelled as an action on activation in the step that follows the transition, or as an action on deactivation in the step preceding the transition.

2) Continuous actions: In IEC 61499, a continuous action (Section III-6) can be implemented with a START/STOP logic. Introducing an additional ECC state ensures that the action is executed as long as the step is active: It is started in the first state and stopped in the second one. The algorithms ensure that the activation/deactivation are performed correctly and safely. The output events are required to update data outputs. As a simple example (Fig. 8), consider activating a digital output. The algorithms will modify the boolean value of the output variable and an output event triggers a FB that handles the hardware access. In GRAFCET, continuous actions can have an additional condition, i.e., they are only executed if the additional condition is fulfilled (Fig. 9). ECC algorithms can have additional if-conditions to control their execution (written e.g. in Structured Text), while sending output events can only be controlled via ECC transitions. The conditional continuous action can be interpreted as a continuous action combined with additional transitions as shown in the right image of Fig. 9. This version mimics the same behavior: the action is executed as long as the boolean condition $c1$ is true and the step S1 is active, and, as soon as $c2$ becomes true, the step S2 is reached. Taking this transformation into account, the Grafcet on the right can be directly translated to an ECC using the previous pattern, resulting in an equivalent ECC as shown in Fig. 10. In the resulting ECC, the exclusive conditions such as $c2$ and $c1\&\overline{c2}$ have been implemented with the priorities of transition evaluation, choosing always the shortest path to the following

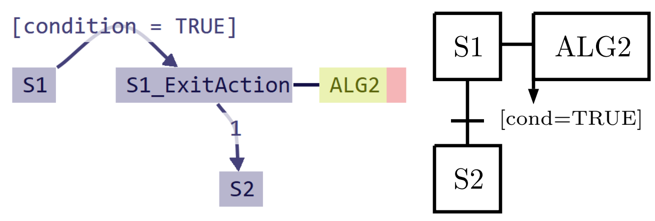

Fig. 6. An action on deactivation was added to an intermediate state, from which S2 is entered immediately.

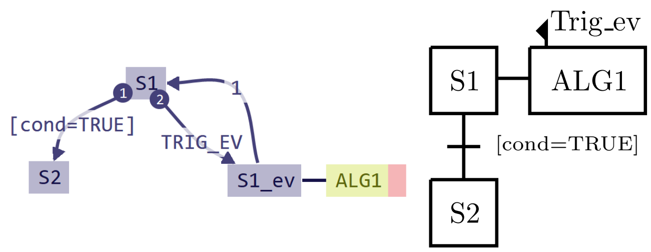

Fig. 7. Action on event is added to S1_ev which is entered when a TRIG_EV event arrives, then S1 is set active again.

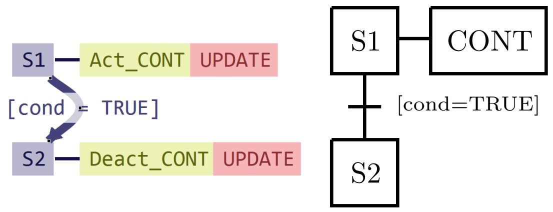

Fig. 8. A continuous action is modelled by two states which are activate and deactivate the continuous action CONT.

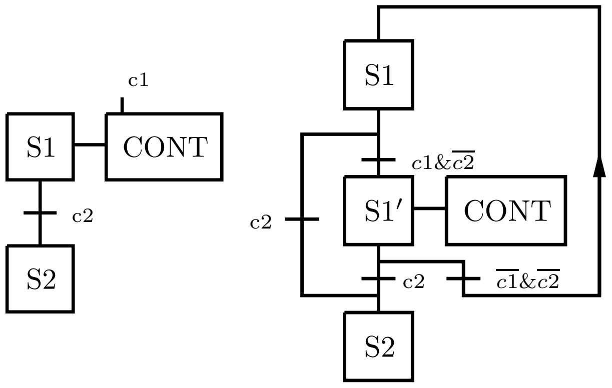

Fig. 9. Functional interpretation of a conditional continuous action (left) as a continuous action with additional transitions (right).

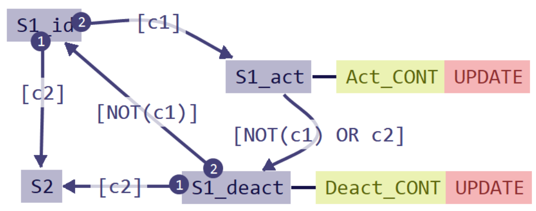

Fig. 10. ECC of a conditional continuous action in GRAFCET such as the example presented in Fig. 9.

state. In addition, the deactivation procedure of the continuous action has been implemented in a single state from which the next state is decided. Continuous actions in GRAFCET are only executed if the step in which they are placed is stable (Section III-4). In Fig. 11, an example of an unstable step with its translation pattern is presented. The unstable step has been modelled with an additional ECC state holding the continuous action in the case that the step is stable. The transition with higher priority bypasses the action execution.

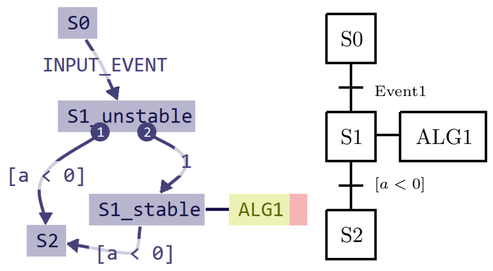

Fig. 11. ECC of an unstable step. The action is bypassed if S1 is unstable because the transition priority was chosen accordingly.

3) Time-triggered conditions: In GRAFCET modelling, conditions that depend on time are very common, and they are used for example in transitions and conditional continuous actions. ECCs do not have an equivalent structure, however, timeout blocks connected to the FB can be used to implement a time-triggered behavior. To reduce the event and data interface of the control FB, the timer FB is connected via an adapter. An adapter extends a FB interface and groups data and events. Consider a simple example where a transition must be traversed 2 seconds after entering the preceding state (Fig. 12). A timer FB is connected to a timer adapter port (a so-called plug) at the interface of the BFB. In the source state of the time-triggered transition, the timer is set and started. The timer FB sends an event as soon as the timeout has been reached and the transition is traversed. Events and data through an adapter are addressed like structured datatypes: the name of the socket is followed by a point and the name of the event or data (e.g. timer.START).

V. Evaluation Example

To exemplify the transformation principles stated above, let us consider the example presented in Annex B of [3], the automatic weighing-mixing. The configuration is presented in Fig. 13. The system consists of two hoppers which contain products A and B. The products are dispensed to a weighing

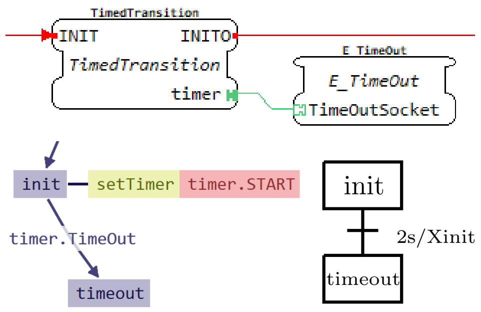

Fig. 12. Implementation pattern for a transition triggered by a time-depending condition. Timeout parameter is set in the algorithm: timer.DT := T#2s;

unit C. Once the desired weight of each product has been obtained, this unit feeds both products into a mixer N, into which a belt also feeds soluble bricks. All the products are mixed in N and, once the mixing process has finished, the mixture is poured, the container of the mixer returns to its original position, and the process awaits to restart.

The standard also proposes several GRAFCET representations of the system's functioning. For the scope of this article, the representation involving macrosteps is specifically interesting. An extended version of that Grafcet will be used as the starting point for implementing a control program.

A. Implementation

Let us consider the expanded Grafcet model presented in [3], Figure B4. The application will be distributed to four devices, one that initiates the sequence, the weighing unit, the belt, and the mixer. First, the distribution principles stated in sections IV-A1 and IV-A2 were applied. They do not alter the behavior of the original Grafcet, but lead to structural changes.

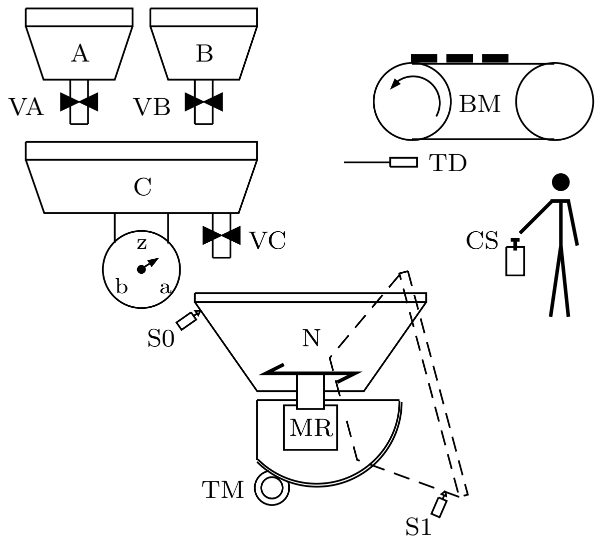

Fig. 13. Automatic weighing-mixing system configuration diagram presented in Annex B in [3].

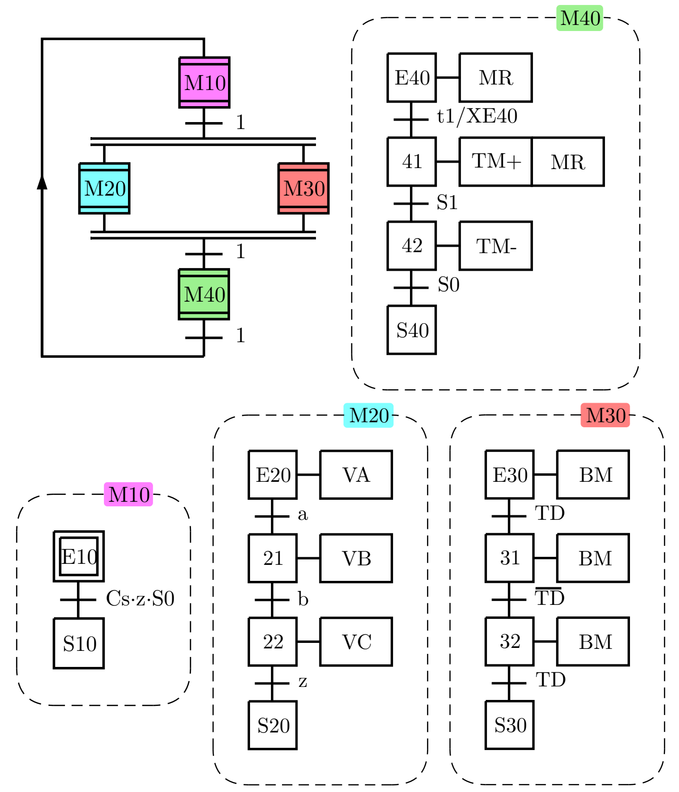

Fig. 14. Grafcet of the described system synthesized and normalized for implementation.

The resulting Grafcet will be translated to IEC 61499 and is presented in Fig. 14.

The application of IV-A1 has four subapplications, which already allows distributing the application. The subapplications are marked in different colors. They consist of a Grafcet chain where only one state is active at a time, so according to IV-A2, each of them can be compacted into a macrostep modelled by a single BFB.

The transitions that connect macrosteps M40 and M10 are resolved by a simple event connection. To resolve the parallelism between M20 and M30 and its posterior synchronization the pattern described in section IV-A3 is applied. In Fig. 15, the FBD resulting from applying the translation patterns refering to the structure is presented. Here the four subapplications, the parallelism, the synchronization and the distribution in different devices can be seen.

As described in IV-A2, the subapplications are composed of a FB network in charge or reading variables, implementing the control algorithms and refreshing the outputs. The only exception is in M10 where variables z and S0 have to be received from their original subapplication. The subapplication that corresponds to M40 is shown in Fig. 16, which contains all the possible elements described above. In this subapplication, as S0 is needed in M10, it is sent along with its event to the subapplication that models M10.

Regarding the ECCs of the control FBs, the principles presented in IV-B2 and IV-B3 are applied, resulting in a

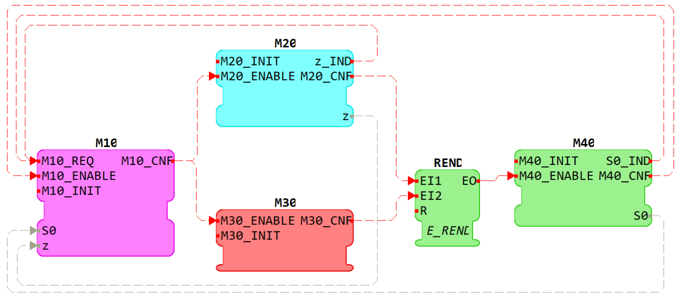

Fig. 15. Control application in 4diac IDE for the proposed Grafcet.

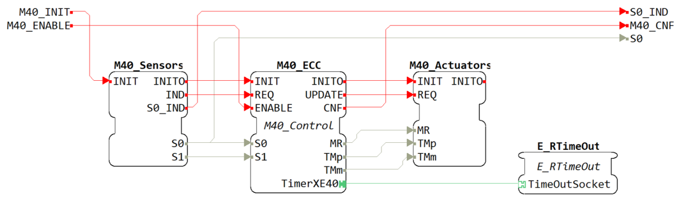

Fig. 16. Subapplication corresponding to the partial GRAFCET M40.

ECC very similar to the original GRAFCET. In Fig. 17, the ECC corresponding to the control FB representing the partial GRAFCET M40 is presented. The algorithms have been condensed to contain both the activation and deactivation of the associated action.

The presented ECC, once it is "ENABLED", will evolve with the calls to the "REQ" event that will refresh the value of the input data. For this example, an event is sent to the REQ socket whenever the input variables change in value. This is done by plugging the IND event from the digital input FBs directly to the REQ socket. The same implementation procedure has been followed for the other partial Grafcets, which is not presented for simpler cases with less elements than the case of M40.

VI. Conclusions and Outlooks

A systematic procedure for implementing control software in IEC 61499 departing from a Grafcet model has been proposed. This work enables engineers to implement Grafcet models that are distributed over several devices while keeping

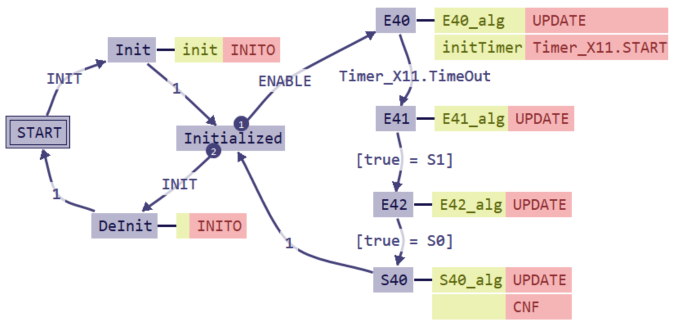

Fig. 17. ECC of the control FB representing the partial GRAFCET M40

the design centralized. This is a key advantage compared to state-of-the-art implementations of Grafcet models.

This work proposes a systematic translation from GRAFCET to IEC 61499, and introduces a variety of translation patterns. These translation patterns are possible because of the similarities between GRAFCET and ECC, which allow modelling most language elements of GRAFCET. Nevertheless, structuring mechanisms such as enclosures or forcing steps cannot be modelled in IEC 61499 and macrostep implementation is limited to simple sequences. An application example where the proposed guidelines are used is presented, illustrating the systematic implementation approach.

This work constitutes an important first step towards a standardized implementation of applications, usually modelled and implemented in a centralized fashion, in the distributed and modular context that IEC 61499 offers. However, several aspects require further investigation. For example, the translation patterns are presented for steps with only one action, when more than one action is present, specifically when they are of different types, some adjustments to the patterns or structural changes to the model need to be done to keep the functionality.

The execution of the algorithms is ruled by the event generation policy. However, this aspect has not been discussed. Several alternatives can be promising in terms of performance depending on the context, to provide data evidence on which policy to apply in each context further work is required.

Despite IEC 61499 being a standard that focuses on distribution, issues derived of this distribution such as the communication delays between modules have not been fully studied, and, depending on the nature of the system, it may play an important role in the global performance.

Finally, investigating how to include the behavior of the structuring mechanisms proposed in Grafcet modelling into IEC 61499 application design could lead to applications which would not only contain a richer hierarchical design than IEC 61131-programs, but also benefit from the distribution configurability that the IEC 61499 standard offers.

Acknowledgments

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

References

[1] M. Brambilla, J. Cabot, and M. Wimmer, *Model-driven software engineering in practice*. Synthesis lectures on software engineering, Morgan & Claypool, 2nd edition ed., 2017.

[2] Object Management Group, "OMG systems modeling language (OMG SysML): Version 1.6," November 2019.

[3] IEC, "IEC 60848: GRAFCET specification language for sequential function charts," tech. rep., International Electrotechnical Commission (IEC), 2002.

[4] D. Harel, "Statecharts: a visual formalism for complex systems," *Science of Computer Programming*, vol. 8, no. 3, pp. 231–274, 1987.

[5] 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.

[6] M. Völter, *DSL engineering: Designing, implementing and using domain-specific languages*. Lexington, KY: CreateSpace Independent Publishing Platform, 2010-2013.

[7] IEC, "IEC 61131 - programmable controllers, part 3: Programming languages," 2013.

[8] V. Vyatkin, "Software engineering in industrial automation: State-of-the-art review," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1234–1249, 2013.

[9] B. Werner, "Object-oriented extensions for IEC 61131-3," *IEEE Industrial Electronics Magazine*, vol. 3, no. 4, pp. 36–39, 2009.

[10] Eclipse 4diac, "Eclipse 4diac - the open source environment for distributed industrial automation and control systems," 2020.

[11] R. David and H. Alla, *Discrete, continuous, and hybrid Petri nets*, vol. 1. Springer, 2005.

[12] G. Frey, "PLC programming for hybrid systems via signal interpreted Petri nets," in *Proc. 4th Int. Conf. Autom. Mix. Process. ADPM*, pp. 189–194, 2000.

[13] S. Klein, X. Weng, G. Frey, J.-J. Lesage, and L. Litz, "Controller design for an FMS using Signal Interpreted Petri Nets and SFC: validation of both descriptions via model-checking," in *Proceedings of the 2002 American Control Conference*, vol. 5, pp. 4141–4146, IEEE, 2002.

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

[15] G. Frey, "Automatic implementation of Petri net based control algorithms on PLC," in *Proceedings of the 2000 American Control Conference. ACC*, vol. 4, pp. 2819–2823, IEEE, 2000.

[16] I. Jimenez, E. Lopez, and A. Ramirez, "Synthesis of ladder diagrams from Petri nets controller models," in *Proceeding of the 2001 IEEE International Symposium on Intelligent Control (ISIC'01)*, pp. 225–230, IEEE, 2001.

[17] M. Wenger, A. Zoitl, C. Sunder, and H. Steininger, "Transformation of IEC 61131-3 to IEC 61499 based on a model driven development approach," in *7th IEEE Int. Conf. on Industrial Informatics, 2009*, (Piscataway, NJ), pp. 715–720, IEEE, 2009.

[18] M. Riedl, C. Diedrich, and F. Naumann, "Sfc inside iec 61499," in *2006 IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 662–666, 2006.

[19] A. Barji, N. Hagge, and B. Wagner, "Comparative study of using CNet, IEC 61499, and statecharts for behavioral models of real-time control applications," in *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 750–757, 2006.

[20] N. Hagge, "Integrating CNet and IEC 61499 function blocks," in *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, pp. 506–509, 2007.

[21] R. Schoop and A. Strelzoft, "Asynchronous and synchronous approaches for programming distributed control systems based on standards," *Control Engineering Practice*, vol. 4, pp. 855–861, jun 1996.

[22] B. Wiesmayr, L. Sonnleithner, and A. Zoitl, "Structuring distributed control applications for adaptability," in *ICPS 2020, Tampere. In Press.*, 2020.