Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 180 (2021) 13–23

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Industry 4.0 and Smart Manufacturing

A GEMMA-GRAFCET Methodology to enable Digital Twin based on Real-Time Coupling

Giacomo Barbieri[a,*], David Andres Gutierrez[b]

[a]*Department of Mechanical Engineering, Universidad de los Andes, Bogotá (Colombia)*
[b]*Xcelgo A/S, Ry (Denmark)*

Abstract

Digital Twin (DT) represents the next wave in modelling, simulation and optimization technology. A key enabling technology of the DT is the real-time coupling of the digital models with the controllers of the physical plant. To implement the DT through real-time coupling, common approaches and vocabularies are desirable for the automation software to facilitate the exchange of information between the PLCs and the digital models. However, the scarce adoption of guidelines and standards within the industrial practice of PLC programming complicates this context. To face this challenge, this paper proposes a methodology for PLC code to standardize the vocabulary and the management of the operational modes of industrial automation systems. The methodology consists of a GEMMA-GRAFCET representation and a Hierarchical Design Pattern. By implementing the approach to a case study, it is demonstrated that the methodology generates a standard interface for the communication between PLCs and digital models within a DT architecture.

Keywords: Digital Twin; PLC; GEMMA; GRAFCET.

1. Introduction

Within the industry 4.0 paradigm, *Digital Twin* (DT) represents the next wave in modelling, simulation and optimization technology [1]. The continuous interaction, communication, and synchronization among the digital models, their physical twin and the external environment pave the way for a '*closed-loop optimization*' able to continuously optimize the physical system [2].

An enabling technology of the DT is the *real-time coupling* of the digital models with the controllers of the physical production plant [3]. Since PLCs (Programmable Logic Controllers) are central controls within the commonly adopted

* Corresponding author. Giacomo Barbieri
E-mail address: g.barbieri@uniandes.edu.co

1877-0509
10.1016/j.procs.2021.01.122

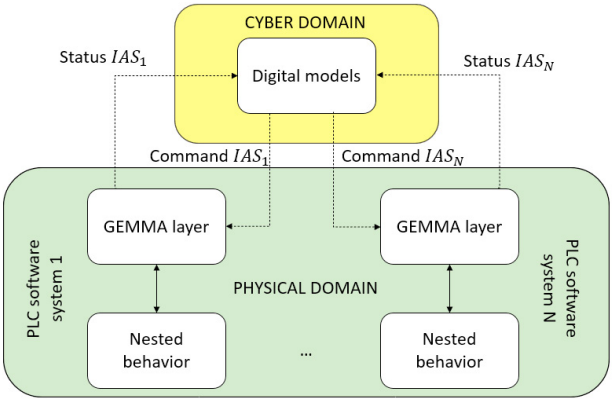


Fig. 1. Proposed DT architecture based on real-time coupling: a standard interface is established for each IAS by adopting the GEMMA guideline.

automation pyramid [4], it is desirable to define standard interfaces to facilitate the exchange of real-time information between the *PLC software* and the digital models.

The *GEMMA guideline* provides a common approach and vocabulary for the management of the operational modes (OMs) of Industrial Automation Systems (IASs) [5]. However, the guideline is not widespread in the current industrial practice of PLC programming [6]. One of the reasons may be that GEMMA is partially specified with respect to the syntax and semantics causing misunderstandings and possible emergent behaviors. Therefore, the first novelty presented in this paper is the enhancement of the GEMMA guideline with the *GRAFCET* standard for the generation of a representation able to completely specify the management of the OMs.

Furthermore, the GEMMA-GRAFCET representation must consist in a one-to-one translation of the PLC software to effectively represent it. Given that, the second novelty of this paper is the introduction of a *Hierarchical Design Pattern* (HDP) for the deployment of PLC software specified in accordance with the proposed GEMMA-GRAFCET representation. As shown in Figure 1, the HDP generates hierarchical PLC software for separating the management of the OMs (common for each IAS) from their nested behavior (specific to each IAS). In this way, each IAS shares the same interface with the digital models facilitating the communication within the DT architecture.

The added value generated from the application of the proposed methodology is the development of a common *approach and vocabulary* for the PLC management of the OMs of IASs. In the context of DT, a standard interface is thus established for the software, facilitating the coupling of the PLCs with the digital models. Given the above, the paper is structured as follows: state of the art is summarized in Section 2, while the proposed GEMMA-GRAFCET representation is described in Section 3. Section 4 illustrates the HDP, and Section 5 applies the whole methodology to a case study. Obtained results are discussed in Section 6, while Section 7 presents the conclusions and future work.

2. State of the Art

In the literature, different formal languages have been proposed for modeling the behavior of IASs [7]. However, the introduced approaches still lack acceptance in the industrial practice of PLC programming since are based on modelling languages which are usually not familiar to the designated users [8].

In response to this, Julius et al. [9] propose to use *GRAFCET* (Graphe Fonctionnel de Commande des Étapes et Transitions) for modeling PLC software, since widely accepted within the automation community [10]. Then, they introduce a systematic approach to automatically translate GRAFCET diagrams into PLC code.

Even if GRAFCET is accepted within the automation community, it does not provide a general approach for the definition and handling of the *Operational Modes* of IASs [11]. Therefore, researchers start looking at guidelines introduced from industrial and academic associations. *GEMMA* (Guide des Modes d'Etude et d'Arrêts Marches [5]) is a checklist which allows to graphically define all the start and stop modes of a system and their evolution. *S88* standard is used specifically to guide the development of batch control processes [12]. *PackML* (Packaging Machine Language) is specialized for the OMs of packaging machinery [13]. Even if one of the authors has experience with

PackML [14], GEMMA is selected within this paper since it is not specific to any special kind of controlled process. However, the approach proposed in this work can be also applied to the other guidelines; see Section 4.

Concerning the conversion of *GEMMA-based specifications* into PLC software, Machado and Seabra propose a methodology in [15]. A high-level GEMMA diagram is used for specifying the system states and their transitions. Then, the functionality that must be implemented within each state is described by means of GRAFCET diagrams. Finally, both the GEMMA and the GRAFCET diagrams are converted into Sequential Function Chart (SFC) Program Organization Units (POUs).

Alvarez et al. [6] develop the '*Methodology for industrial Automation systems*' (MeiA) supported with an implementation framework. This methodology combines GEMMA, UML use case diagrams and GRAFCET for assisting the control practitioners during the analysis, design and coding phases. OMs are identified by means of the GEMMA guideline. Use case diagrams are used for defining the actors that take part into the OMs. A set of GRAFCET templates assists in the design of the OMs specified in accordance with the GEMMA guideline. Finally, Model-Driven Engineering (MDE) is adopted to convert GRAFCET diagrams into SFC POUs.

With respect to [15, 6], a different approach is proposed in this work for the PLC management of the OMs of IASs using the GEMMA guideline, since: (i) they use the GRAFCET to specify the nested behavior of the GEMMA states, but not to fix the lack of formalism of the guideline; (ii) they convert GRAFCET diagrams into SFC POUs limiting code readability, since long code is generated when large GRAFCET diagrams are implemented.

3. GEMMA-GRAFCET Representation

GEMMA is a guide for performing a systematic search of the possible states of an IAS from a control perspective. The states are grouped into three families of start and stop modes: 'F family' for the operation procedures, 'A family' related to the stop procedures, and 'D family' concerning the failure procedures. GEMMA offers a *graphical notation* for specifying the sequential behavior of IASs. However, it is partially specified with respect to the syntax and semantics causing misunderstandings and possible *emergent behaviors*. For instance, the guideline does not allow to specify the initial state or whether a transition can occur even if the nested behavior of the state has not been completed yet.

To avoid the development of emergent behaviors, we propose to implement the GEMMA guideline through a *formal modeling language*. The selected modeling language must fulfill the following *functional requirements* to implement and to semantically specify the GEMMA guideline:

- *Sequential behavior*: system behavior must be defined as a sequence of states and transitions connected by directed links
- *Transition condition*: a Boolean condition must be set on each transition to evaluate when the transition to the subsequent state can take place
- *Transition priority*: when a state has more than one outgoing transition, a priority must be set to specify the order in which conditions are evaluated
- *Alternative branches*: the modeling language must allow the definition of mutually exclusive transitions since the guideline establishes that only one state can be active at a time
- *Initial State*: one initial state must be selected to define from which state the system starts to operate
- *Hierarchical structure*: the nested behavior of each 'GEMMA state' must be separated from the one specified at the GEMMA hierarchical layer (Fig. 1)
- *Exit state*: it must be possible to specify whether a transition at the GEMMA hierarchical layer can occur even if the nested behavior has not been completed yet
- *Composite state*: when an emergency occurs, system must move from any of the states of the 'F family' towards the 'Emergency stop' state independently from the active state of the nested behavior.

Along with these functional requirements, the following *non-functional requirements* must be fulfilled in order to be accepted within the automation community: (i) *Automation modeling language*: the selected modeling language must be familiar to the control practitioners; (ii) *Consistency*: the implemented representation must consist in a one-to-one translation of the GEMMA graphical notation.

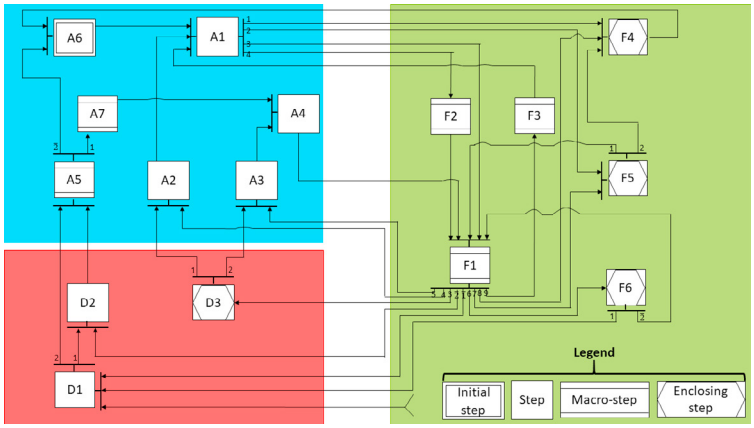


Fig. 2. GEMMA-GRAFCET representation proposed for specifying the OMs of IAS.

To semantically specify the GEMMA guideline, we propose to use the GRAFCET modeling language. GRAFCET was introduced in 1982 and later became an *international standard* (IEC 60848 [16]). GRAFCET is familiar to the control practitioners since has become an inherent part of the professional education of control technicians [17].

Next, it is demonstrated how the GRAFCET standard fulfills most of the identified functional requirements:

- *Sequential behavior* → GRAFCET is a graphical language that describes the behavior of sequential systems by means of steps with associated actions, transitions and oriented lines
- *Transition condition* → transitions are defined with associated conditions
- *Transition priority* → GRAFCET does not explicitly allow the specification of the priority of the transitions
- *Alternative branches* → alternative branches can be represented by means of divergences and convergences in ”or”
- *Initial State* → a GRAFCET model always has at least one initial step
- *Hierarchical structure* → partial diagrams can be enclosed into macro-steps or enclosing steps encouraging a hierarchical architecture for modeling complex processes
- *Exit state* → the use of a macro-step indicates that the outgoing transitions can only fire when the exit step of the nested behavior is active. Whereas, the outgoing transitions of an enclosing step can fire independently from the active step of the nested behavior. However, GRAFCET does not allow to ’customize’ the macro-step / enclosing step behavior based on the considered outgoing transition
- *Composite state* → the use of an enclosing step can move the system from any of the state of the ’F family’ towards the ’Emergency stop’ state, independently from the active step of the nested behavior.

The GRAFCET model of the GEMMA guideline is shown in Figure 2. Steps, initial step, macro-steps and enclosing steps have been arbitrary placed since the representation is not relative to any specific case study. The ’or divergence’ is used when a step has more than one outgoing transition, while the ’or convergence’ when a step has more than one incoming transition. Colored areas are placed on the background of the steps to visually subdivide the states into the three families of start and stop modes defined within the GEMMA guideline.

Finally, the following exceptions to the standard have been implemented to fulfill the functional requirements not achieved with the use of GRAFCET:

1. *Transition priority*: when a state has more than one outgoing transition, a number is placed on each transition to indicate its priority; i.e. the order in which conditions are evaluated within the code. This strategy is taken from the SFC specifications defined by the PLCopen organization[1]

[1] https://www.plcopen.org/sites/default/files/downloads/sfc_textual.pdf

2. *Exit state*: a negation operator can be placed on the priority number. This operator changes the behavior of the composite state with respect to the one illustrated in the representation – when the 'negated' transition is evaluated. For instance, F1 in Figure 2 is represented as a macro-step. The $\overline{1}$ in the outgoing transition to the D1 state indicates that F1 must be considered as an enclosing step – when this transition is evaluated. This strategy enables the customization of the macro/enclosing step behavior, based on the considered outgoing transition
3. *Composite state*: the 'F family' acts as an enclosing step. However, GRAFCET does not allow to visualize the nested behavior on the same layer of the super-ordinate step; i.e. inside the box of the enclosing step. Therefore, an outgoing transition towards the 'Emergency stop' (D1 state in Figure 2) is placed within the green background area. When the code is developed, control practitioners must consider this area as an enclosing step.

Comparing the GEMMA guideline (see [5]) with the introduced GEMMA-GRAFCET representation (Fig. 2), it can be noticed that the representation consists in a *one-to-one translation* of the GEMMA graphical notation. Therefore, the automation modeling language and consistency non-functional requirements have been achieved. Finally, the lack of formalism of the GEMMA guideline has been fixed by enhancing the guideline with the GRAFCET standard and by introducing three exceptions to it. The resulting GEMMA-GRAFCET representation can now be utilized to completely specify the management of the OMs of IASs.

4. Hierarchical Design Pattern

To effectively mimic the PLC software, the GEMMA-GRAFCET representation must consist in a one-to-one translation of the code. The implementation of the GEMMA guideline through the GRAFCET modeling language imposes the following *functional requirements* for the PLC code:

- *Active state*: the philosophy of the GEMMA guideline is to have only one active state at a time
- *Initial state*: one initial state must be selected to define from which state the system starts to operate
- *Reset transition*: GRAFCET imposes that the nested behavior is initialized each time the super-ordinate state is entered
- *Entry and continuous behavior*: with respect to the state-machine formalism [7], GRAFCET imposes that the states do not implement exit behaviors
- *Macro-step and enclosing step*: the use of a macro-step indicates that the outgoing transitions can only fire when the exit step of the nested behavior is active. Whereas, the outgoing transitions of an enclosing step can fire independently from the active step of the nested behavior
- *Vertical coordination*: synchronization in between the GEMMA hierarchical layer and the nested behavior.

Then, the following *non-functional requirements* must be fulfilled to assist humans to design, verify, supervise and maintain the PLC software [18]: (i) *Hierarchical code*: code must be decomposed into separated hierarchical layers; (ii) *Readable and maintainable code* must be generated; (iii) *Modular code*: each element must be independent and must communicate with standard interfaces.

Next, the *Hierarchical Design Pattern* able to fulfill the identified functional and non-functional requirements is introduced for the generation of PLC software defined in accordance with the GEMMA-GRAFCET representation proposed in Section 3. The design pattern is shown through an illustrative example in which the specifications indicated in the left-hand side of Figure 3 are implemented. These specifications mimic a GEMMA diagram that includes initial macro-step S1, macro-step S2, enclosing step S3, and S4 and S5 steps. An 'or divergence' is implemented as outgoing transition of the initial step S1, in which T2 has higher priority than T1. Furthermore, T1 has a negation operator indicating that S1 must be considered as an enclosing step – when T1 is evaluated.

The *software architecture* is first defined. The implemented PLC code must be hierarchically structured for separating the nested behavior of each 'GEMMA state' from the one specified at the GEMMA hierarchical layer; see Fig. 1. As [9], we decide to adopt *Function Blocks* (FBs) for hierarchically structuring the code. One *program* is generated for implementing the behavior of the GEMMA layer, and one FB for each 'GEMMA state' that contains a nested behavior; i.e. macro-steps and enclosing steps. Only the `GEMMA` program is scheduled through a periodic task, while the execution of the nested behavior is invoked from the 'GEMMA active state' by calling the corresponding FB.

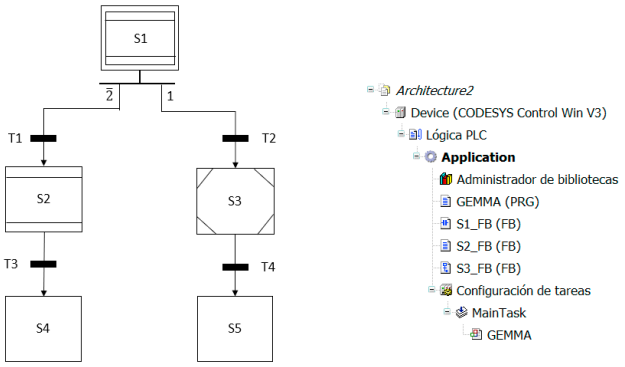


Fig. 3. Illustrative example: a) GEMMA-GRAFCET specifications; b) software architecture and scheduling strategy.

Through this strategy, *vertical coordination and synchronization* is granted since the nested behavior is supervised and coordinated from the GEMMA hierarchical layer.

In the illustrative example, a program is created for the GEMMA diagram, while FBs for the states that have a nested behavior; i.e. S1, S2 and S3. `GEMMA` is the only scheduled POU (right-hand side of Figure 3), while `S1_FB`, `S2_FB` and `S3_FB` are invoked from the `GEMMA` program. In the proposed design pattern, the control practitioner can choose any programming language for the implementation of the function blocks, while *Structured Text* (ST) must be adopted for the `GEMMA` program. This choice facilitates the interaction in between 'old and new generations' of practitioners, since they are used to program in different languages.

Next, the *GEMMA hierarchical layer* is converted into *PLC code*. ST is selected since more flexible and compact with respect to the other programming languages. Moreover, ST provides better readability and more intuitive programming to today's users that are usually familiar with at least one C-like language. The philosophy of the GEMMA guideline is to have only one active state at a time. Design patterns have been proposed for the conversion into ST of state-machines that only have one active state for each hierarchical layer [19]. A widespread pattern is to use a scalar `State` variable as a discriminator of a *switch statement*; i.e. `CASE .. OF` in ST. Transitions are evaluated – according to their priority – by means of `IF .. THEN` constructs and if a transition is fireable, the new state is assigned. In this way, each state remains active for at least one PLC cycle. To implement the proposed scheduling strategy, each 'GEMMA state' invokes the FB that contains its nested behavior. Entry actions are performed using an `Entry` flag and an `IF .. THEN` construct that resets the flag, once the entry action has been performed. Concerning the *interaction* in between the GEMMA layer and the *nested behavior*, the code must handle: (i) reset transitions; (ii) macro-steps and enclosing steps. To manage reset transitions and macro/enclosing steps, two *flags* are used (i.e. `Initialization` and `Complete`), respectively declared as input and output variables of each FB. `Initialization` is utilized for implementing the reset transition behavior. `Initialization` is set on the entry action of each 'GEMMA state' and reset on the continuous action. Whereas, the `Complete` flag is used to implement the *macro-step* behavior. The outgoing transitions of a macro-step have an additional `AND` condition consisting in the `Complete` flag. This flag is set within the exit step of the nested behavior and reset during its reset transition. Whereas, an *enclosing step* does not have the `Complete` flag condition on the outgoing transitions.

The left-hand side of Figure 4 shows the code written within the `GEMMA` program for the implementation of the GEMMA-GRAFCET representation of the illustrative example. On the *variable declaration* section, `State` and `Entry` are declared, and the FBs are instantiated. `State` is initialized with the value of 1 in accordance with the specifications of Figure 3, while `Entry` as `TRUE` to activate the token for the execution of the entry action. Then, a `CASE .. OF` structure is used for modeling the *state evolution*. The `Initialization` flag of the corresponding FB is set as entry action of the state that contains a nested behavior; i.e. macro-steps and enclosing steps. Then, the FB that implements the corresponding nested functionality is invoked as continuous action. Finally, *outgoing transitions* are evaluated in accordance with their priority and eventual negation operators. For instance, `T2` is evaluated before `T1` since has higher priority. Moreover, `T1` transition makes the macro-step `S1` behave as an enclosing step, since has a negation operator; see left-hand side of Figure 3. Concerning the implementation of macro-steps and enclosing steps, the code of states `S2` and `S3` can be compared (left-hand side of Figure 4). It can be noticed that the outgoing

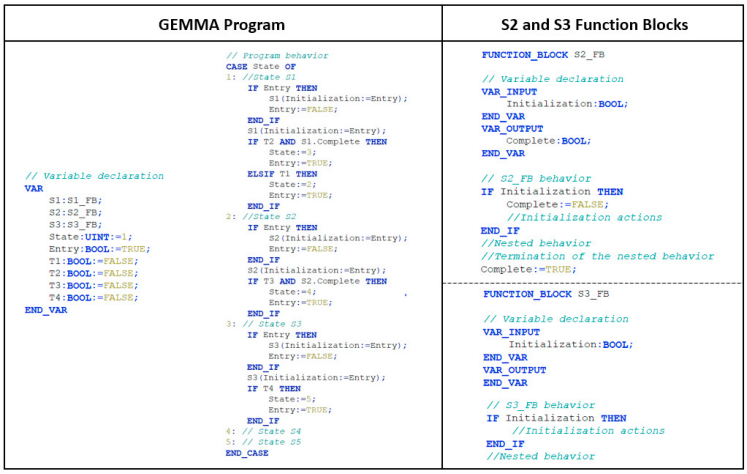


Fig. 4. PLC code of the illustrative example: a) GEMMA program; b) S2 and S3 function blocks. Within the GEMMA program, it can be noticed that: (i) T2 is evaluated before T1, since has higher priority; (ii) due to the negation operator of transition T1, S1 acts as an enclosing step when T1 is evaluated, while as a macro-step when T2 is evaluated.

transition of S2 has an additional AND condition given by the Complete flag. This flag allows the implementation of a *macro-step* behavior, while its absence the one of an *enclosing step*.

Finally, the code generated within the S2_FB and S3_FB is shown in right-hand side of Figure 4. The Initialization flag is utilized for initializing the behavior as soon as the state is entered, and its actions include the reset of the Complete flag. Then, the nested behavior is performed, and the Complete flag is set once the nested behavior has been completed. S3_FB is characterized with the same code but without the setting of the Complete flag, since an outgoing transition of an enclosing state can fire independently from the active step of the nested behavior.

Next, the HDP fulfilment of the identified functional and non-functional requirements is verified. The requirement is recalled on the left-hand side, while the strategy for its fulfillment is placed on the right-hand side:

- *Functional requirements*:
  - *Active state* → CASE .. OF structure
  - *Initial state* → initial value of the State variable
  - *Reset transition* → Initialization flag
  - *Entry and continuous behavior* → Entry flag
  - *Macro-step and enclosing step* → FBs and Complete flag
  - *Vertical coordination* → scheduling strategy
- *Non-functional requirements*:
  - *Hierarchical code* → software architecture: separation of the nested behavior of each 'GEMMA state' from the one specified at the GEMMA hierarchical layer
  - *Readable and maintainable code* → in comparison to [9], the defined pattern is simpler and more intuitive since FBs do not need methods for their operation and management. In comparison to [15, 6], ST is used for the implementation instead of SFC making the code more synthetic and readable. Finally, the code is characterized by repetitive patterns that may become familiar to the control practitioners and in future be automatized through MDE
  - *Modular code* → use of FBs to encapsulate the nested behavior of each 'GEMMA state' and standard interface in between the GEMMA program and the nested behavior.

In summary, the proposed HDP fulfills all the identified functional and non-functional requirements generating a one-to-one correspondence between the PLC software and the GEMMA-GRAFCET representation. Furthermore, the design pattern may be applied also to PackML and ISA88, since these guidelines have only one active state at a time and are characterized by hierarchical behavior.

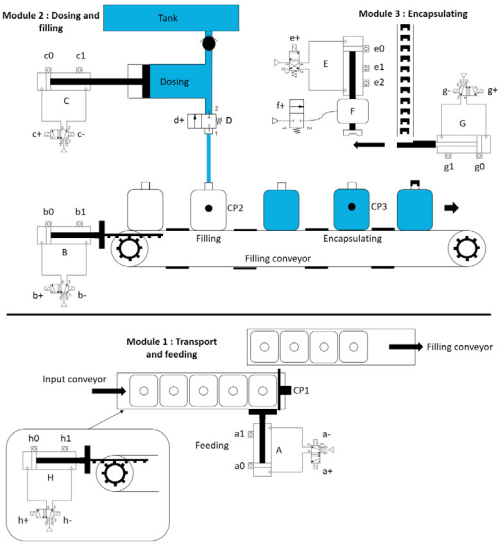


Fig. 5. Schematic representation of the filling and encapsulating machine.

It can be noticed that due to the selected scheduling strategy, the nested behaviors (i.e. FBs) have the same periodicity of the GEMMA program. This is acceptable since the GEMMA guideline is used for the management of the OMs and does not need to monitor and control 'fast dynamics'. For instance, programs with different scheduling strategies should be implemented for failure detection.

5. Case Study

A case study was developed for demonstrating that the proposed methodology enables the DT through real-time coupling. An automatic machine for *filling and encapsulating bottles* was selected (Figure 5) since has been used in several GEMMA-based researches; see [11, 20, 15]. The machine consists of three stations: (i) transport and feeding; (ii) dosing and filling; (iii) encapsulating. Two stepping slat conveyors allow the simultaneous operation of the three stations. Extension of pneumatic cylinders H and B respectively determines the incremental advance of the input and filling conveyors. The *transport and feeding station* is constituted by pneumatic cylinder A that is responsible for the feeding of the bottles from the input to the filling conveyor. The *dosing and filling station* is composed by a volumetric dispenser actuated from pneumatic cylinder C, and by an on/off valve (D) used to open and close the liquid supply. The *encapsulating station* has a pneumatic cylinder (G) to feed the cap that is recollected and released on the bottle through the movement of cylinder E. Vacuum pump F is utilized to grab and hold the cap. Finally, magnetic limit switches are used to signal the position of the pneumatic cylinders and light barrier sensors to indicate the presence of a bottle on each station.

First, the specifications of the filling and encapsulating machine were identified using the illustrated *GEMMA-GRAFCET representation* (Fig. 6). After the identification of the GEMMA hierarchical layer, the *nested behavior* of each state was defined. The nested behavior is made available to the reader[2] but is not illustrated since does not constitute the focus of this work. Then, the *HDP* presented in Section 4 was applied for the conversion of the specifications into PLC code written into the CoDeSys[3] programming environment. The *software architecture* was developed by generating one program for implementing the behavior of the GEMMA diagram, and one FB for each macro-step and enclosing step; see left hand-side of Figure 7. The `GEMMA` program is scheduled with a periodic task, while the FBs are invoked from the `GEMMA` program as shown in the center part of Figure 7 for the `F2` state. Finally,

[2] https://www.researchgate.net/publication/344568872_Filling_machine_software_specifications
[3] https://www.codesys.com/

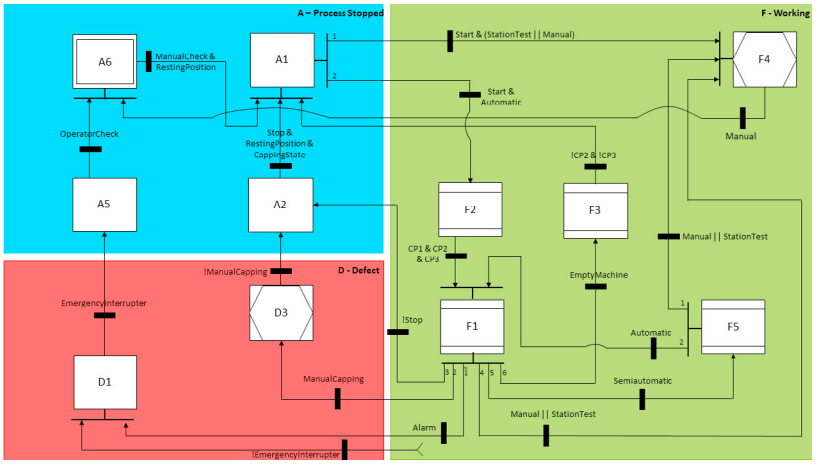
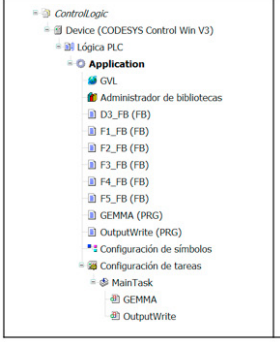


Fig. 6. GEMMA-GRAFCET representation of the filling and encapsulating machine.

Software architecture



GEMMA Program

```
CASE State OF
1: //A1 State
⋮
9: //F2 State
    //Entry action
    IF Entry THEN
        F2(Initialization:=Entry);
        Entry:=FALSE;
    END_IF
    //Continuos action
    F2(Initialization:=Entry);
    //Transtions and exit action
    IF NOT GVL.Emergency THEN
        State:=14; //Transition to D1
        Entry:=TRUE;
    ELSIF F2.Complete AND GVL.CP1 AND GVL.CP2 AND GVL.CP3 THEN
        State:=6; //Transition to F1
        Entry:=TRUE;
        GVL.L_F2:=FALSE;
    END_IF
10: //F3 State
⋮
END_CASE
```

F2 Function Block

```
//Reset transition
IF Initialization THEN
    State:=1;
    Complete:=FALSE;
END_IF

//Nested behavior
CASE State OF
1:
⋮
3:
    IF GVL.CP1 AND NOT GVL.CP2 THEN
        State:=4;
    ELSIF GVL.CP1 AND GVL.CP2 AND NOT GVL.CP3 THEN
        State:=6;
        Sub1:=7;
        Sub2:=9;
    ELSIF NOT GVL.CP1 THEN
        State:=1;
    ELSIF GVL.CP1 AND GVL.CP2 AND GVL.CP3 THEN
        State:=11;
    END_IF
⋮
11: //Final step
    Complete:=TRUE;
END_CASE
```

Fig. 7. Design pattern illustrated for the F2 state. It can be noticed that the transition to the D1 state has the highest priority – since acts at the 'F family' level – and makes the F2 state behave as an enclosing step.

the nested behavior of the macro-steps and the enclosing steps was generated. The image on the right-hand side of Figure 7 illustrates part of the code implemented for the F2 FB.

After writing the PLC code, the *DT architecture* shown in the left-hand side of Figure 8 was built. A kinematic model of the filling and encapsulating machine was built in *Experior*[4] (center part of Fig. 8) and connected with CoDeSys for the implementation of a *Virtual Commissioning* (VC) simulation [21]. This simulation acted as the physical system within the DT architecture. Finally, an HMI (Human Machine Interface) was built in *Simulink*[5] (right-hand side of Fig. 8) and acted as the digital model in the cyber space.

6. Results and Discussion

After the development of the simulation models, the DT architecture was run making Experior, CoDeSys and Simulink communicate through the *OPC protocol*. Videos showing use cases of the machine functioning are made available to the reader[6].

[4] https://xcelgo.com/
[5] https://mathworks.com/products/simulink.html
[6] https://www.youtube.com/watch?v=9v48dQM44RA&list=PLXnpmcz3YbS-YpaXKUeQKXyK4PorRlDzx

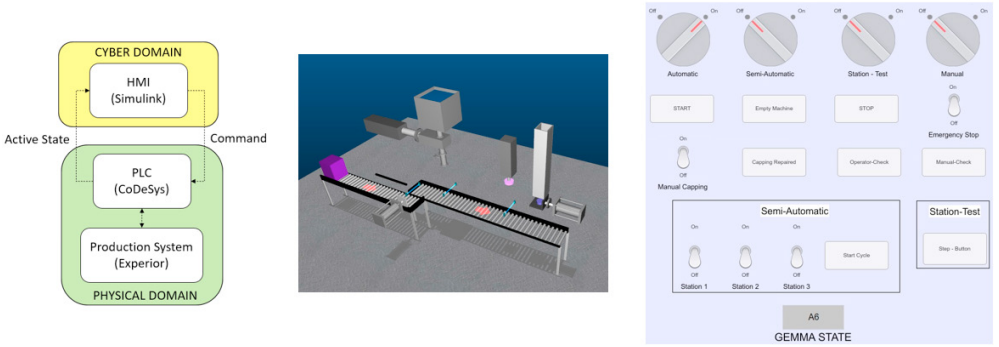


Fig. 8. Methodology validation: a) DT architecture; b) Experior model of the filling and encapsulating machine; c) Simulink HMI.

The right-hand side of Figure 8 shows that the Simulink HMI receives the 'GEMMA active state' from the CoDeSys PLC and controls the machine by means of buttons. By making few students – with knowledge in the GEMMA guideline but not involved in this work – control the simulation, we noticed that the use of the *GEMMA-GRAFCET representation* generated a common understanding of the machine status, and they were able to control the system without a detailed knowledge of the functioning of the nested behaviors. Finally, the *Hierarchical Design Pattern* allowed the implementation of the intended machine behavior and consisted in a one-to-one mapping of the GEMMA-GRAFCET representation, reaching the objective for which it had been proposed.

We are conscious that a DT involves a physical system and has at least one simulation model that monitors and optimizes the system through a bidirectional communication in between the physical and the cyber space [22]. However, we chose to develop an *HMI* on the cyber space (and not a digital model) since the focus of this work was the definition of a standard interface for the management of the OMs of IASs, and not to optimize a specific application.

In summary, the GEMMA-GRAFCET methodology introduced in this work and tested in a lab case study has the premises to be transferable also to industrial production systems. In fact, the methodology allows to: (i) completely specify the OMs of IASs following the GEMMA guideline – without the occurrence of emergent behaviors; (ii) generate a hierarchical, modular, readable and maintainable PLC code which consists in a one-to-one translation of the GEMMA-GRAFCET specifications. In the context of DT, a standard interface is thus established for the software, facilitating the coupling of the PLCs with the digital models.

7. Conclusion and Future Work

To enable the DT through real-time coupling, PLCs and digital models must be interfaced. Few guidelines have been introduced for managing the OMs of IASs and may act as standard interfaces within the DT communication. However, these guidelines are currently not widespread in the industrial practice of PLC programming. Considering that one of the reasons may be the lack of formalism of the available guidelines, this work proposes a methodology to develop a *common approach and vocabulary* for the PLC management of the OMs of IASs using the GEMMA guideline. The methodology consists of: (i) *GEMMA-GRAFCET representation* for specifying the management of the OMs of IASs without the occurrence of emergent behaviors; (ii) *Hierarchical Design Pattern* for the generation of hierarchical PLC code from specifications expressed in accordance with the proposed GEMMA-GRAFCET representation. After applying the methodology to a case study, it has been demonstrated how the approach generates a standard interface that – in the context of DT through real-time coupling – can facilitate the exchange of information between the PLCs and the digital models.

The proposed *GEMMA-GRAFCET methodology* constitutes a preliminary concept that in the future should be further validated and improved. Some future works identified are:

- *Methodology Validation*: the GEMMA-GRAFCET methodology has been applied to a virtual lab case study and validated based on the feedback of designated users. To better validate it, the proposed approach will be implemented in a physical production system and quantitative metrics will be computed

- *Model Driven Engineering*: on the one hand, the proposed HDP is characterized by repetitive patterns that in future may be automated through the MDE approach. On the other hand, a graphical tool may be developed to facilitate the definition of software specifications based on the GEMMA-GRAFCET representation. A 'default' template containing all the possible 'GEMMA states' and transitions should be available to the control practitioners. Then, they should be able to change the type of state (initial step / step / macro-step / enclosing step), and to delete the states and transitions that are not necessary for the case study. Finally, they should specify the condition, priority and eventual negation operator of the transitions before the generation of the PLC code
- *Virtual Commissioning in Digital Twin*: the role of VC with respect to DT will be validated with a proper DT case study. In this work, the digital model was implemented through an HMI. In future work, a digital model will be developed to monitor and optimize the virtual production plant. This case study will allow to validate the role of the VC as a tool for the design and verification of the DT functionality before its implementation.

References

[1] Rosen R., Von Wichert G., Lo G., and Bettenhausen K. D. (2015) "About the importance of autonomy and digital twins for the future of manufacturing." *IFAC-PapersOnLine* **48** (3): 567–572.

[2] Barricelli B. R., Casiraghi E., and Fogli D. (2019) "A Survey on Digital Twin: Definitions, Characteristics, Applications, and Design Implications." *IEEE Access* **7**: 167653–167671.

[3] Biesinger F., and Weyrich M. (2019) "The Facets of Digital Twins in Production and the Automotive Industry." *23rd International Conference on Mechatronics Technology (ICMT)*

[4] Monostori L. (2014) "Cyber-physical production systems: Roots, expectations and R&D challenges." *Procedia Cirp*, **17**: 9–13.

[5] ADEPA. (1981) "GEMMA (Guide d'Étude des Modes de Marches et d'Arrets)." *Agence nationale pour le Developpment de la Production Automatisée*.

[6] Alvarez M. L., Sarachaga I., Burgos A., Estévez, E., and Marcos M. (2016) "A methodological approach to model-driven design and development of automation systems." *IEEE Transactions on Automation Science and Engineering* **15** (1): 67–79.

[7] Friedenthal S., Moore A., and Steiner R. (2014) "A practical guide to SysML: the systems modeling language", *Morgan Kaufmann (eds)*

[8] Vogel-Heuser B., Diedrich C., Fay A., Jeschke S., Kowalewski S., and Wollschlaeger M. (2014) "Challenges for software engineering in automation." *Journal of Software Engineering and Applications*.

[9] Julius R., Schurenberg M., Schumacher F., and Fay A. (2017) "Transformation of GRAFCET to PLC code including hierarchical structures." *Control Engineering Practice* **64**: 173–194.

[10] Arzén K. E. (1994) "Grafcet for intelligent supervisory control applications." *Automatica* **30** (10): 1513–1525.

[11] Cloutier G., and Paques J. J. (1988) 'GEMMA, the complementary tool of the Grafcet." *Fourth Annual Canadian Conference Proceedings in Programmable Control and Automation Technology Conference and Exhibition*

[12] Parshall J., and Lamb L. (1999) "Applying S88: Batch Control from a User's Perspective", *ISA*

[13] Arens D., Hopfgartner T., Jensen T., Lamping M., Pieper M., and Seger D. (2006) "Packaging Machine Language V3. 0 Mode & States Definition Document." *OMAC Motion for Packaging Working Group*.

[14] Barbieri G., Battilani N., and Fantuzzi C. (2015) "A PackML-based Design Pattern for Modular PLC Code." *IFAC-PapersOnLine* **48** (10): 178–183.

[15] Machado J. M., and Seabra E. (2010) 'A systematized approach to obtain dependable controllers specifications." *ABCM Symposium Series in Mechatronics*

[16] International Electrotechnical Commission. (2002) "Grafcet specification language for sequential function charts." *IEC 60848*.

[17] Marichal R. L., and González E. J. (2014) "ULLSIMGRAF: An educational tool with syntax control for Grafcet notation." *Computer Applications in Engineering Education* **22** (4): 669–677.

[18] Dotoli M., Fay A., Miśkowicz M., and Seatzu C. (2019) "An overview of current technologies and emerging trends in factory automation." *International Journal of Production Research* **57** (15-16): 5047–5067.

[19] Bonfe M., Fantuzzi C., and Secchi, C. (2013) "Design patterns for model-based automation software design and implementation." *Control Engineering Practice* **21** (11): 1608–1619.

[20] Balcells J., Romeral, J. L., and Martínez J. L. R. (1997) "Autómatas programables", *Marcombo*

[21] Lee C. G., and Park S. C. (2014) "Survey on the virtual commissioning of manufacturing systems." *Journal of Computational Design and Engineering* **1** (3): 213–222.

[22] Kritzinger W., Karner M., Traar G., Henjes J., and Sihn W. (2018) "Digital twin in manufacturing: A categorical literature review and classification." *FAC-PapersOnLine* **51** (11): 1016–1022.