Hierarchical-Modular Control for the prototype of a Flexible Manufacturing Cell

A. I. Hernández-Silva [*] **, E. Aranda-Bricaire** [*]

[*] *Departamento de Ingeniería Eléctrica, Sección de Mecatrónica, CINVESTAV, AP. 14-740, 07000 Ciudad de México(e-mail: aldo.hernandez@cinvestav.mx, earanda@cinvestav.mx).*

Abstract: This work presents the implementation of a Hierarchical-Modular Control architecture for the prototype of a Flexible Manufacturing Cell. Finite-state automata are used for the modelling of the cell. Supervisory Control Theory is employed to restrict its operation and control architecture for product recipes definition. Given monolithic supervisors that guarantee a safe operation of the prototype, product recipes are added in order to resemble a Flexible Manufacturing Cell capable of producing different products with minimal modifications of the control architecture.

Keywords: flexible manufacturing cell, discrete-event systems, supervisory control, automata, hierarchical control, modular control.

1 INTRODUCTION

Nowadays, manufacturing industry is facing a fast evolution, due to an increasing demand and retrenchment in delivery time. To meet the new production requirements imposed by this philosophy, most manufactures are obligated to optimize their processes. One way to do so, is by implementing the idea of Flexible Manufacturing Systems (FMS), where a manufacturing cell is conceived to vary its operation according to a given product recipe. A FMS consists in several automated workstations, interconnected via automatic systems for material handling and distribution (Groover 2000). These systems can be modeled as discrete event systems. Discrete Event Systems (DES), are those whose state space is finite and whose state can only change as a result of asynchronously occurring instantaneous events over time (Cassandras and Lafortune 2009). Industrial processes fall among this category. DES are modelled using formalisms such as automata, petri nets, etc. The issue of achieving a specific behavior is addressed by Supervisory Control Theory (SCT).

In SCT, a large system can be abstracted by modelling its various subsystems and imposing an accepted behavior on it (Afzalian and Noorbakhsh and Wohnham 2010). One major difficulty arises when the control is synthetized because the state-space grows exponentially with the number of subsystems resulting in a computational problem for designers and industrial implementation (De Queiroz and Cury 2002).

This paper continues the work done in (Rodríguez-Zúñiga 2018), where safety operation of a *Flexible Manufacturing Cell* (FMC) is achieved using specifications. As it was shown in the paper previously mentioned, the resulting supervisory control become complex in terms of the state space dimension. Therefore, in the present work a hierarchical architecture is used to add new operational features additionally to the already programmed for a safety-mode operation. These new features are known as *product recipes* (Hernández-Silva 2019) which are defined and synthetized in a higher control level. As a consequence, a two-level hierarchical control architecture controls the operation of the FMC. To achieve a synergetic operation between safety supervisory control and high-level product recipes, hierarchical consistency and non-blocking properties are used.

2 PRELIMINARIES

In this section, standard notation and basic concepts of automata theory (Rosen 2012) and supervisor control theory (Cassandras and Lafortune 2009) (Wohnam 2017) are summarized.

2.1 Languages and Finite-state Automata

A finite state automaton (FSA) is an intuitive and natural description of a DES, it has a finite number of states and transits between these states on the occurrence of events. State transition diagram is the simplest way to present the idea of automaton (Fig. 1). Formally, an automaton G is a six-tuple:

$$G=\{X, \Sigma, f, \Gamma, x_0, X_m\},$$

Where X is the *state set*, Σ is the *event set (or alphabet)*, $f : X \times E \rightarrow X$ is the *(partially defined)*

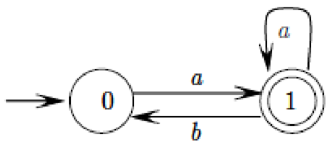


Fig. 1. State transition diagram of an automaton.

transition function, Γ is the *active event function* (set of all events σ for which $f(x,\sigma)$ is defined), x_0 is the initial state, $X_m \subseteq X$ is the set of marked states. A language L defined over an alphabet Σ is a set of finite-length strings formed from events in Σ. The *language generated* by an automaton G depicts all the strings that can be generated by the system. It is defined by $L(G) := \{ s \in E^* | \ f(x_0, s) \text{ is defined}\}$. The *marked language* L_m, represents all the desired strings in a system, it is defined by $L_m(G) := \{ s \in L(G) | \, f(x_0, s) \in X_m\}$.

Some important operations over languages are presented next. For a finite event set Σ, the set of all finite strings over Σ is denoted by Σ^* (Kleene-closure), in the languages context $L^* := \{\epsilon\} \cup L \cup LL \cup \cdots$. *Concatenation* of two strings is represented by $s_1 s_2 \in \Sigma^*$. The *empty string* $\epsilon \in \Sigma^*$ is the identity element of concatenation, i.e., $s\epsilon = \epsilon s = s, \forall s \in \Sigma^*$. The *prefix-closure* of a language is defined by $\overline{L} := \{s \in \Sigma^* : (\exists t \in \Sigma^*)[st \in L]\}$. The *natural projection* from a larger set of events Σ_l, to a smaller set of events $\Sigma_s \subset \Sigma_l$, is defined by $P : \Sigma_l^* \to \Sigma_s^*$

$$\begin{aligned} P(\epsilon) &:= \epsilon \\ P(\sigma) &:= \sigma \quad \text{if} \quad \sigma \in \Sigma_s \\ P(\sigma) &:= \epsilon \quad \text{if} \quad \sigma \in \Sigma_l \setminus \Sigma_s \end{aligned}$$

In other words, this operation takes a string formed from the larger event set and erases events in it that do not belong to the smaller event set.

Sometimes two (or more) automata are required to operate together with the intention to interact. This joint behaviour is modelled by *Parallel composition*, denoted by $||$. For two automata G_1 and G_2 it is defined as follows:
$G_1 || G_2 := X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f, \Gamma_{1||2}, (x_{01}, x_{02}), X_{m1} \times X_{m2}$

Where
$f((x_1, x_2), \sigma) :=$

$$\begin{cases} (f_1(x_1,\sigma), f_2(x_2,\sigma)) & si\ \sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ (f_1(x_1,\sigma), x_2) & si\ \sigma \in \Gamma_1(x_1) \setminus \Sigma_2 \\ x_1, f_2(x_2,\sigma)) & si\ \sigma \in \Gamma_2(x_2) \setminus \Sigma_1 \\ undefined & otherwise \end{cases}$$

2.2 Supervisory control

A DES modelled by an automaton G, describes a behavior represented by its generated language $L(G)$. The event set can be decomposed as $\Sigma = \Sigma_c \cup \Sigma_{uc}$, where Σ_c is the set of *controllable events* and Σ_{uc} is the set of *uncontrollable events*. Sometimes the operation of the prototype is hazardous, i.e., it is not satisfactory and must be restricted. To do so, W. M. Wonham and P. J. Ramadage developed the celebrated Supervisory Control Theory. The paradigm of this theory is based on the existence of a *supervisor S*, which controls the transition function of G, in the sense that controllable events of G can be dynamically enabled or disabled. The expression S/G denotes the operation of system G under the action of supervisor S. A controlled system is said to be *non-blocking* if $L(S/G) = \overline{L_m(S/G)}$. This property outstanding since, if it is not fulfilled, a system cannot complete a task. Additionally, a supervisor must satisfy the *controllability* condition, stated as follows: for a given language K such that $L(S/G) = \overline{K}$, it is said to be controllable with respect to a system G and a set of uncontrollable events $\Sigma_{uc} \subseteq \Sigma$, if $\overline{K}\Sigma_{uc} \cap L(H) \subseteq \overline{K}$. A supervisor fulfills both controllability and non-blocking conditions if it meets the following theorem (Cassandras and Lafortune 2009).

Theorem 1. Non-blocking Controllability Theorem. Given a DES G with $\Sigma_{uc} \subseteq \Sigma$. There exists a non-blocking supervisor S for G such that $L_m(S/G) = K$ and $L(S/G) = \overline{K}$ iff the two following conditions are held:

$$\text{Controlability} : \overline{K}\Sigma_{uc} \cap L(H) \subseteq \overline{K} \qquad (1)$$

$$L_m(G) - \text{closure} : K = \overline{K} \cap \mathrm{L}_m(G) \qquad (2)$$

2.3 Modular supervisory control

Using parallel composition between all the components of a system and all the specifications of the desired performance, one obtains the model of the close-loop system (*monolithic supervisor*). During the supervisory control synthesis, the major challenge in the area of DES arises because the state space of the system grows exponentially with respect to the number of components. Contrary to this approach, the *modular supervisory control* exploits the modularity of specifications and the decentralized structure of composite plants. According to this, a modular supervisor is synthetized for each specification over a local plant. Preserving the *non-conflicting* property ensures that the local modular supervisors have the same accomplishment as monolithic supervisors (Fabian 2006). The non-conflict property of two supervisors is verified if:

$$\overline{L_m(S_1/G) \cap L_m(S_2/G)} = \overline{L_m(S_1/G)} \cap \overline{L_m(S_2/G)} \quad (3)$$

2.4 Hierarchical control

Hierarchical control addresses the state space explosion by introducing a two-level structure (Fig. 2). At the lowest level, a closed-loop system is established between G and S^l (Wong and Wonham 1996), where $\mathrm{Con}^{\mathrm{l}}$ is the control action from supervisor and $\mathrm{Inf}^{\mathrm{l}}$ is the feedback information from the system. The highest level is constructed via *Hierarchical abstraction* denoted by $\mathrm{Inf}^{\mathrm{lh}}$, which performs a natural projection over the plant G, resulting in a smaller high-level model of the plant G^h. Then, the high-level supervisor synthesis is computed using G^h, obtaining a high-level closed-loop system, where $\mathrm{Inf}^{\mathrm{h}}$ is the feedback

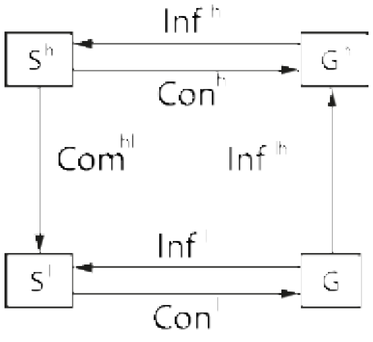


Fig. 2. Hierarchical control architecture

information channel and Con$^{\mathrm{h}}$ is the control action. Finally, Com$^{\mathrm{hl}}$ represents the communication between the supervisors of the two levels.

2.4.1 Hierarchical abstraction of decentralized systems

Hierarchical architectures reduce the complexity of supervisor synthesis by only considering the relevant information from the plant G, so the high-level plant G^h has always less or equal number of states compared to G (Schmidt and Marchand 2005). When there is the necessity of adding more subsystems to the original plant, the process of redesign becomes computationally expensive as the low-level plant G must be composed considering all subsystems. After this, a low-level supervisor is synthetized and then, the process on Hierarchical abstraction is performed. To prevent this computationally-expensive process, a plant G is considered as a *Decentralized control system*, which consists of several smaller components with their own functionality. These smaller components interact to exhibit the behavior of overall system. It is possible to project the decentralized subsystems to the higher level first and then, form the total high-level control system via parallel composition (Schmidt and Moor 2008). To achieve this, the following definitions are presented. First, a *high-level alphabet* Σ^h, is defined based on the events needed to obtain a desired behavior . *Decentralized high-level alphabets* are defined as $\Sigma_i^h := \Sigma^h \cap \Sigma^i$, where Σ^i represents the alphabet of each module. *Decentralized natural projections* are defined as $p_i^{dec} : \Sigma_i^* \to \left(\Sigma_i^h\right)^*$ for $i = 1, \ldots, n$, where n is the total number of decentralized subsystems. Using the previous definitions, the following is stated: $G^h = ||_{i=1}^n p_i^{dec}(G_i)$. Finally, in order to compute a supervisor S^l, it must satisfy the non-blocking property and needs to be *hierarchical consistent* (Zhong and Wonham 1990). Hierarchical consistency property is achieved if a decentralized low-level specification $K_i \subseteq \Sigma_i^*$ meets the following criteria (Schmidt and Moor 2008):

$$p_i^h \left(L_m \left(S^h/G^h\right)\right) \subseteq p_i^{dec} \left(K_i\right) \tag{4}$$

$$K_i \text{ controllable w.r.t } G_i \textit{ and } \Sigma_{i,uc} \tag{5}$$

And the non-blocking property is fulfilled if:

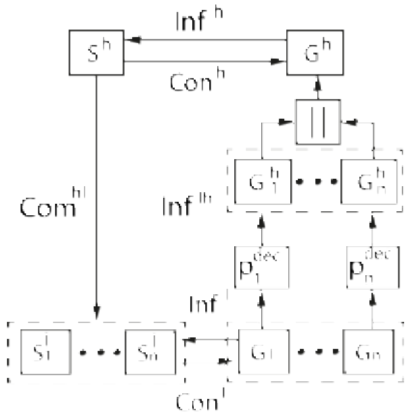


Fig. 3. Hierarchical decentralized architecture

$$\overline{L_m \left(S_i^l/G_i\right)} = L_m \left(S_i^l/G_i\right) \tag{6}$$

The hierarchical and decentralized architecture is depicted in Figure 3.

3 FLEXIBLE MANUFACTURING CELL

The rest of the paper focuses on a laboratory case study, namely, the flexible manufacturing cell made by the German brand Fischertechnik availabke in the Discrete-Event Systems laboratory of Electric Engineering Department at CINVESTAV. The cell consists in 4 modules that represent different processes resembling an actual manufacturing industry installation (Fig. 4). The modules are: 1) Pneumatic Center, 2) Machining Center, 3) Pressing Center, 4) 3 DOF Robotic Arm. The cell is designed to transport a piece by conveyor belts throughout the stations resembling a manufacturing process. Previous results on this manufacturing cell (Rodríguez-Zúñiga 2018)(Rodríguez-Zúñiga and Aranda-Bricaire 2018)(Hernández-Silva 2019) show the modelling process of the four modules using automata. Four monolithic supervisors synthetized from safety and procedure specifications to follow a desired behavior of each module were presentend as well.

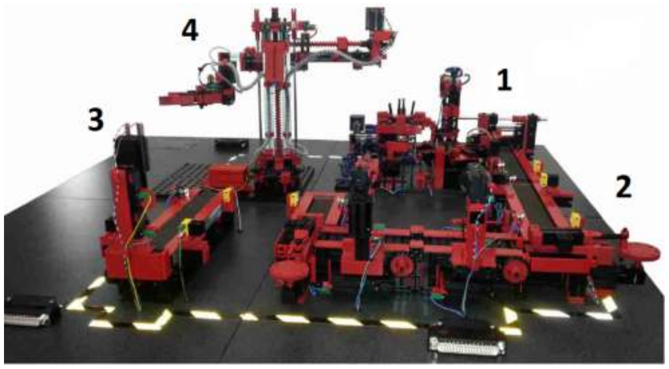


Fig. 4. Flexible manufacturing cell

In Table 1, condensed information about pneumatic, machining, pressing and robotic arm modules is presented.

Table 1. Modules information

Module	**Plant**	**Supervisor**	**Prop. 1**	**Prop. 2**
Pneumatic	(16,64)	(7,32)	Yes	Yes
Machining	(256,2048)	(16,160)	Yes	Yes
Pressing	(6,14)	(6,6)	Yes	Yes
Robotic arm	(12,32)	(14,14)	Yes	Yes

In that Table, parentheses denote $(states, transitions)$, property 1 stands for controllability and property 2 stands for non-blocking. To compute complex automata operations in next sections, the software package TCT was used. This software was developed by Renyuan Zhang and W. M. Wonham (Wonham 2017).

4 HIERARCHICAL-MODULAR CONTROL

The main purpose of hierarchical-modular approach is to provide extra information to a given structure, in this case, four monolithic supervisors and four plants are already defined and modeled. The extra information to be added will be *product recipes*. For a given recipe, the four process machines of the FMC are considered to be enabled or disabled. Considering this, the high level alphabet Σ^h is defined based on the controllable and uncontrollable events of: press machine 1 $\Sigma_{PM1} = \{121, 120\}$, milling machine $\Sigma_{MM} = \{211, 210\}$, lathe machine $\Sigma_{LM} = \{221, 220\}$ and press machine 2 $\Sigma_{PM2} = \{321, 320\}$ as follows:

$$\Sigma^h=\{120,\ 121,\ 210,\ 211,\ 220,\ 221,\ 320,\ 321\}$$

In Table 2, the desired recipes are defined based on the controllable events of the high level alphabet $\Sigma_c^h = \{121,\ 211,\ 221,\ 321\}$.

Table 2. Product recipes

Recipe	**Press 1 (121)**	**Milling (211)**	**Lathe (221)**	**Press 2 (321)**
R_0	No	No	No	No
R_1	Yes	No	No	No
R_2	No	Yes	No	No
R_3	Yes	Yes	No	No
R_4	No	No	Yes	No
R_5	Yes	No	Yes	No
R_6	No	Yes	Yes	No
R_7	Yes	Yes	Yes	No
R_8	No	No	No	Yes
R_9	Yes	No	No	Yes
R_{10}	No	Yes	No	Yes
R_{11}	Yes	Yes	No	Yes
R_{12}	No	No	Yes	Yes
R_{13}	Yes	No	Yes	Yes
R_{14}	No	Yes	Yes	Yes
R_{15}	Yes	Yes	Yes	Yes

From Table 2, recipes are grouped into 5 cases:
1) one machine operates, recipes: $R_1,\ R_2,\ R_4$ and R_8,
2) two machines operate: $R_3,\ R_5,\ R_6,\ R_9,\ R_{10}$ and R_{12},
3) three machines operate, recipes: $R_7,\ R_{11},\ R_{13}$ and R_{14},
4) four machines operate, recipe: R_{15},
5) no machine operates, recipe: R_0.

The four process machines consider for the recipes (press machine 1, milling machine, lathe machine and press machine 2) are part of only three modules in FMC: Pneumatic, Machining and Pressing. For this reason, to perform the hierarchial abstraction three decentralized high-level alphabets are defined: $\Sigma_1^h = \{120,\ 121\}$, $\Sigma_2^h = \{210,\ 211,\ 220,\ 221\}$ and $\Sigma_3^h = \{320,\ 321\}$. With such definitions, the four process machines can be abstracted from the model of each module using the decentralized natural projections. Finally, the resulting automatan from each projection, are grouped using parallel composition to obtain the high-level plant G^h. The previous process simplifies the computational complexity for supervisor synthesis as it uses a reduced model of the plant and by focusing in specific events of the whole FMC. In this case, G^h consists of 16 states and 64 transitions.

4.1 Supervisor synthesis

To achieve the desired behavior, high-level specifications are introducced. Generic automata are used to represent recipes where two (Fig. 5) and three (Fig. 6) machines operate. The main action of high-level specifications, is to impose a sequence for machines operation. For example, manufacturing process a is followed by manufacturing process b. Therefore, the specifications for recipes R_3, R_5, R_9, R_{10} and R_{12} are represented using the following automaton (Fig. 5).

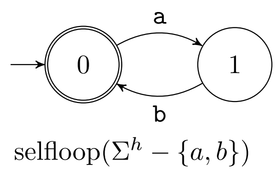


Fig. 5. Specification for recipes R_3, R_5, R_9, R_{10} and R_{12}

Where $a, b \in \Sigma_c^h$. The specifications for recipes R_7, R_{11}, R_{13} and R_{14} are represented using the following automaton (Fig. 6).

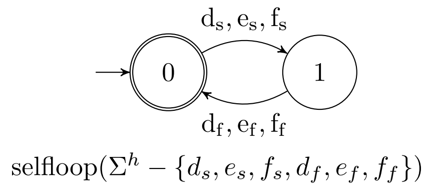


Fig. 6. Specification for recipes R_7, R_{11}, R_{13} and R_{14}

Where $d_s, e_s, f_s \in \Sigma_c^h$, $\mathrm{d_f, e_f, f_f} \in \Sigma_{uc}^h = \{120, 210, 220, 320\}$, subscript s and f stand for the start and finish of a manufacturing process in a machine, respectively. The previous specification states that one manufacturing process must finish before another starts. The specification used for recipe R_{15} is presented in Figure 7.

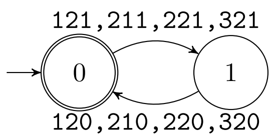


Fig. 7. Specification for recipe R_{15}

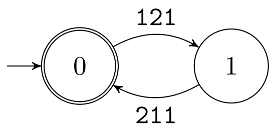
Similarly to the specification shown in Figure 6, the specification for recipe R_{15} enables only one manufacturing process at a time, but in this case, the four manufacturing processes are available. To obtain a simplified supervisor for the previous recipe especifications, the following TCT instructions were used for each recipe.

$$SCR_i = \text{supcon}(G^h, ER_i) \tag{7}$$

$$CR_i = \text{condat}(G^h, SCR_i) \tag{8}$$

$$SR_i = \text{supreduce}(G^h, SCR_i, CR_i) \tag{9}$$

With $i = 3, 5, 7, 9, 10, 11, 12, 13, 14, 15$. If these reduced supervisors meet the hierarchical consistency, non-blocking and controlability properties, they will be considered as an acceptable high-level supervisors. To exemplify hierarchical consistency for supervisor SR_3 (Fig. 8), it was evaluated using (4). The result is shown in Table 3.



selfloop $(120, 210, 220, 221, 320, 321)$

Fig. 8. Reduced supervisor SR_3

Table 3. Herarchical consistency for SR_3

i	$p_i^h\left(L_m\left(S^h/G^h\right)\right)$	$p_i^{dec}(K_i)$
1	$\{120, 121\}^*$	$\{120, 121\}^*$
2	$\{210, 211, 220, 221\}^*$	$\{210, 211, 220, 221\}^*$
3	$\{320, 321\}^*$	$\{320, 321\}^*$

Where monolithic specification K_i for module i is obtained by using parallel composition between all the specifications for module i. The recipe supervisors SR_i with $i = 3, 5, 7, 9, 10, 11, 12, 13, 14, 15$ were evaluated, as in Table 3, and they fulfilled the hierarchical consistency property (4). Controllability and non-blocking properties were verified using TCT instructions: $SR_icn = \text{condat}\left(G^h, SR_i\right)$ and $SR_inb = \text{trim}(SR_i)$. The results of evaluating hierarchical consistency, controllability and non-blocking for each reduced supervisor, are summarized in the Table 4.

Table 4. High-level supervisors properties

Supervisor	Hierar. consis.	Non-blocking	Controllable
SR_3	Yes	Yes	Yes
SR_5	Yes	Yes	Yes
SR_7	Yes	Yes	Yes
SR_9	Yes	Yes	Yes
SR_{10}	Yes	Yes	Yes
SR_{11}	Yes	Yes	Yes
SR_{12}	Yes	Yes	Yes
SR_{13}	Yes	Yes	Yes
SR_{14}	Yes	Yes	Yes
SR_{15}	Yes	Yes	Yes

The recipes: R_0, R_1, R_2, R_4, R_6 and R_8 are not yet implemented. To do so, a modular supervisor approach was proposed as follows: if two previously synthetized high-level supervisors are used in a high-level modular architecture, the desired behaviour by a non-implemented recipe can be achieved. Table 5 is used to illustrate the method used:

Table 5. High-level modular design process

Supervisor	Press 1 (121)	Milling (211)	Lathe (221)	Press 2 (321)
SR_{12}	No	No	Yes	Yes
SR_5	Yes	No	Yes	No
$\boldsymbol{SR_4}$	No	No	Yes	No

The interpretation is as follows: recipe SR_4 has to disable events $121, 211$ and 321, if the high-level supervisors SR_{12} and SR_5 work modularly, the joint action disables the desired events. Using this high-level modular method, supervisors for the next recipes are obtained: $SR_0 = \text{mod}(SR_{12}, SR_3)$, $SR_1 = \text{mod}(SR_3, SR_9)$, $SR_2 = \text{mod}(SR_3, SR_{10})$, $SR_4 = \text{mod}(SR_5, SR_{12})$, $SR_6 = \text{mod}(SR_7, SR_{14})$ and $SR_8 = \text{mod}(SR_9, SR_{12})$. Where $\text{mod}(X, Y)$ stands for the modular architecture using X and Y high-level supervisors. Since modular approach is used, non-conflict property (Fabian 2006) must be verified. To do so, TCT instruction called **nonconflict** is used. This instruction evaluates if two languajes are conflicting which causes modular architecture failure but, if it returns **"nonconflicting"** message, modularity is preserved. Also, hierarchical consistency, controllability and non-blocking properties must be verified. In this modular case, the supervisors involved must be joined using parallel composition and then such properties tested. In the Table 6 results on this evaluations are summarized.

Table 6. High-level modular supervisors

Modular Supervisor	Hierar. consis.	Non-conflict	Non-blocking	Controllable
SR_0	Yes	Yes	Yes	Yes
SR_1	Yes	Yes	Yes	Yes
SR_2	Yes	Yes	Yes	Yes
SR_4	Yes	Yes	Yes	Yes
SR_6	Yes	Yes	Yes	Yes
SR_8	Yes	Yes	Yes	Yes

Low-level supervisors disable controllable events from the incoming string according to safety specifications. On the other hand, high-level supervisors act on the resulting string, first checking if the string contains events from the high-level alphabet and then, disabling controllable events according to a desired recipe. If the resulting string after low-level supervisor does not include events from the high-level alphabet, this string reaches and acts directly on the plant.

5 IMPLEMENTATION

To test the control architecture, it was translated to ladder language (LAD) following a standard equivalence between finite-state automata and LAD instructions (Silva and De Queiroz 2010). The disabling nature of the controllable events by supervisor must be preserved in LAD, so disabling operation was carried out by adding extra contacts (De Queiroz and Cury 2002).

The hardware used was a Siemens S7-1200 PLC which enables to use TIA Portal software. In this software, its interface has two main windows, one showing all the blocks programmed (project tree) and another showing the current block open (block interface). The Hierarchical-Modular approach exhibits easiness at testing the recipes programmed. The user opens the main block, then he can select amongst all recipes, which one will be uploaded to the PLC by simply changing one block (Fig. 10). In the following URL, the FMC executing the hierarchical modular control architecture for recipe three is presented: https://bit.ly/2XTdrVk.

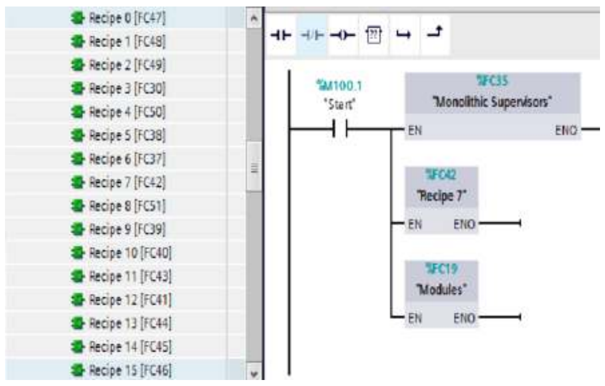


Fig. 9. Hierarchical-Modular architecture test

6 CONCLUSION

The main objective of this paper was to present a method to simplify the process of adding extra information to a controlled system. This method, uses a hierarchical-modular architecture to introduce new processes in a FMC with no safety-risks. Meaning that there is a significant reduction in both computational and programming effort.

7 REFERENCES

Afzalian, A. and Noorbakhsh, S.M. and Wohnham, W.M. (2010). Discrete-event supervisory control for under-load tap-changing transformers (ULTC): from synthesis to PLC implementation.

Cassandras, C.G. and Lafortune, S. (2009). *Introduction to discrete event systems.* Springer Science & Bussiness Media.

De Queiroz, M.H. and Cury, J.E. (2002). Synthesis and implementation of local modular supervisory control for a manufacturing cell. *In Discrete Event Systems, 2002. Proceedings. Sixth International Workshop on,* 377-382. IEEE.

Fabian, M. (2006). *Discrete Event Systems: lecture notes.* Control and automation laboratory, Chalmers University of Technology.

Groover, M.P. (2000). *Automation, Production Systems, and Computer-integrated Manufacturing.* Prentice Hall.

Hernández-Silva, A.I. (2019). *Control Jerárquico-Modular de un prototipo de celda de manufactura flexible.* M. Sc. Thesis, Depto. de Ingeniería Eléctrica CINVESTAV.

Rodríguez-Zúñiga, J.A. (2018). *Modelado y control supervisor de un prototipo de celda de manufactura flexible.* M. Sc. Thesis, Depto. de Ingeniería Eléctrica CINVESTAV.

Rodríguez-Zúñiga, J.A. and Aranda-Bricaire, E. (2018). Modelado y control supervisor de un prototipo de celda de manufactura flexible. *Memorias del Congreso Nacional de Control Automático,* vol. 1, 195-200.

Rosen, K.H. (2012). *Discrete mathematics and its applications.* Mc Graw Hill.

Schmidt, K. and Marchand H. (2005). Modular and Decentralized Supervisory Control of Concurrent Discrete Event Systems Using Reduced System Models. *Proc. Workshop Discrete Event Syst. (WODES),* Ann Arbor, 149-154.

Schmidt, K. and Moor, T. (2008). Nonblocking Hierarchical Control of Decentralized Discrete Event Systems. *IEEE Trans. Automat. Control,* vol. 53, no. 10, 2252-2265.

Silva, Y.G. and De Queiroz, M.H. (2010). Formal synthesis, simulation and automatic code generation of supervisory control for a manufacturing cell. *ABCM Symposium Series in Mechatronics,* vol. 4, 418-426.

Wong, K.C. and Wonham, W.M. (1996). Hierarchical control of discrete event systems. *Discrete Event Dyn. Syst.: Theory Appl.,* vol 6, no. 3, 241-273.

Wonham, W.M. (2017). *Notes on Control of Discrete Events Systems.* Department of Electrical Engineering, University of Toronto, Canada.

Zhong, H. and Wonham, W.M. (1990). On the consistency of hierarchical supervision in discrete-event systems: A state-based approach. *IEEE Trans. Automat. Control,* vol. 35, no. 10, 1125-1134.