Int J Adv Manuf Technol (2010) 48:663–669
DOI 10.1007/s00170-009-2317-z

ORIGINAL ARTICLE

Applying a software framework for supervisory control of a PLC-based flexible manufacturing systems

Belen Curto Diego · Vidal Moreno Rodilla · Carlos Fernandez Carames · Anibal Chehayeb Moran · Raul Alves Santos

Received: 17 March 2009 / Accepted: 15 September 2009 / Published online: 27 October 2009

Abstract In this paper, a software framework for a distributed flexible manufacturing system is presented where, as a main design goal, we consider that the functionalities are offered to other components with no effects on the robustness of the system and where security specifications are tackled without the flexibility losses. In this way, the use of the supervisory control theory (SCT) is considered. Also, a methodology that makes it easier to implement these functionalities at a PLC-based system is presented. A real application of the developed framework is presented, and the implementation is done over an industrial-size device, so the technical feasibility of the methodology is proved. As the framework considers the services offered by the device, it can be used at different manufacturing tasks, so the approach can be applied to any kind of industrial process unit.

Keywords Flexible manufacturing systems · Supervisory control theory · Distributed systems

1 Introduction

Due to the market evolution, the manufacturing industry suffers some pressures in order to reduce product prices, to increase the model complexity, and to support a model proliferation [1]. To achieve these goals, it is necessary to accomplish shorter production cycles and lower manufacturing costs. More flexible and intelligent workcells are needed in order to find a competitive manufacturing process. Thus, as a part of a computer integrated manufacturing (CIM) goal, flexible manufacturing systems (FMS) reached an increase of flexibility, from the point of view of the hardware, when robotic manipulators were included. Current robotic workcell systems have limited functionality and flexibility and low levels of intelligence due to a programmable logic controller (PLC) that acts as a central controller of a cell operation sequence. From a software engineering point of view, problems concerned with the integration and reusability, maintenance, operational costs, and so on also appear.

As a solution, in [6], the use of a distributed architecture is proposed, where more intelligent software components (clients like expert systems, task planners, fault detection elements that run over powerful computers) can offer capabilities that really increment the flexibility and intelligence of the cell. At this proposal, the PLC does not assume the role of activity sequencer in order to perform a task, so it offers services (e.g., move to a given point or move at a given speed for a robot or conveyor belt) to other software components, so they can use the provided funcionalities. In this way, changes at the task specification can be done in a flexible way.

When the first tests of our distributed architecture were realized over a real plant with industrial devices, many wrong situations appear because the system becomes blocked due to an erroneous sequence of service invocations. Evidently, it is necessary to avoid these

B. Curto Diego · V. Moreno Rodilla (✉) ·
C. Fernandez Carames · A. Chehayeb Moran ·
R. Alves Santos
Dept. Computers and Automation,
University of Salamanca, Madrid, Spain
e-mail: vmoreno@abedul.usal.es, control@abedul.usal.es

blocked situations. As a theoretical solution, the use of the supervisory control theory (SCT) [10], which was proposed initially by Ramadge and Wonham, is considered in this work. The SCT has received a special attention from the academic environment in such a way that it has reached a great evolution. Nevertheless, actually just a few applications in the industrial environment that apply the SCT can be found because mainly a complete modelization and implementation of the system (which includes all the hardware and software elements) is needed. Some of the most representative ones can be found at [5, 7, 8], and [3]. The first one describes the way to design a control system for an assembling line, whereas the others are focused on the implementation of a concrete task on a PLC. In most of them, the controller formally obtained using the SCT is restricted to reach a particular goal with a concrete task. In [8], if a modification of the task is made, it is necessary to perform a complete redesign and implementation of the controller.

In [7], a methodology to implement a supervisor over an educational plant is presented. Firstly, they solve an important problem related with PLC programming: the avalanche effect. Secondly, they consider a descomposition of the controller into local controllers with their own specifications. They show that, even when a simple plant is considered, a monolithic supervisor is not tackled. Finally, they propose a method to create the modular supervisors so the problems related to the model size are solved. A change in the task does not require completely computing the supervisor. But it can be considered as a centralized proposal that prevents introducing intelligent components into the global system. Also, the devices of the plant have a laboratory scale, whereas the devices of the industrial system (as the device we have used in this work) have more complex behavior models.

In this work, it is proposed to design the local controller for each device taking into account the services that it provides, with no consideration on the complete task that has to be done because its definition can be dynamically modified. It is clearly related to the modularity approach of [7].

In [5], when the restriction of the plant behavior has to be done, two kinds of specifications can be distinguished: security specifications and progress specifications. The first are considered to prevent that no plant section takes undesirable actions and the second are concerned with reaching the completion of the task. Security specifications are divided into two classes: firstly, those that affect a unique plant element, and secondly, those that involve several elements.

In this work, the key element will be the development and use of a software framework to apply the supervisory control where it will be necessary to model and to implement the set of services and the set of specifications. The second one involves the local security for the device behavior in such a way that if, in a given moment, the attainment of a device service puts in danger the correct performance of the device, the service invocation will be disabled. The global security specifications, which depend of the configuration plant, and the progress specifications, which depend on the task, must be entrusted to a higher-level component related to the task planning software. If a change that affects the production process appears, it will only be necessary to change the sequence in the service invocations that are carried out by the task planner. The device software will remain unaltered and it would not be necessary to make any effort with this software development. So, in [3], a framework that would allow the integration of expert knowledge (using fuzzy rule sets) could be used at higher levels to reach determined performance measures at these levels.

The rest of the paper is organized as follows: in Section 2, the SCT is revised. Next, the main relevant components of the proposed framework will be presented. In Section 4, in order to prove the validity of our proposal, we will present a case of study where a real industrial element will be considered. The model of the plant and its behavior will be shown, so we can present the corresponding use of the framework to implement the controller. Finally, the main conclusions are presented.

2 Modeling discrete event systems

The SCT of discrete event systems is based on the use of automata and formal language models. Under these models, the main interest is on the order in which the different events occur. In this way, a plant [9] is assumed to be the generator of these events. The behavior of the plant model is described by event trajectories over the finite event alphabet Σ. These event trajectories can be thought of as strings over Σ, so $L \subseteq \Sigma^*$ represents the set of those event strings that describe the behavior of the plant. The SCT restricts the behavior of a plant G by temporarily disabling certain events that can be created by Σ, so the goal is that the plant cannot create undesired or illegal event chains in $L(G)$. In the following, a few basic definitions will be posed.

2.1 Plant model

The plant G will be modelled by the deterministic finite state automaton (DFSA):

$$G = (X, \Sigma_G, f_G, \Gamma_G, x_0, X_m),$$

where X is the set of states; Σ_G is the finite set of events over G; $f_G : X \times \Sigma_G \to X$ is the state transition function; $\Gamma_G : X \to 2^{\Sigma_G}$ is the active event function; $x_0 \in X$ is the initial state; and $X_m \subseteq X$ is the set of marked states, which represent the completion of a certain task or a set of tasks. ${\Sigma_G}^*$ is used to denote the set of all finite-length strings over Σ_G, including the empty string ϵ. f_G can be extended from $X \times \Sigma_G$ to the domain $X \times {\Sigma_G}^*$ by means of recursion: $f_G(x, \epsilon) = x$, $f_G(x, se) = f_G(f_G(x, s), e)$ for $x \in X$, $e \in \Sigma_G$ and $s \in {\Sigma_G}^*$. The generated language and marked language are defined, respectively, as $L(G) = \{s \in {\Sigma_G}^* : f(x_0, s) \text{ is defined}\}$ and $L_m(G) = \{s \in L(G) : f(x_0, s) \in X_m\}$.

The SCT distinguishes between two kinds of events: the controllable set (Σ_c), which can be disabled, and the non-controllable event set (Σ_{nc}), which cannot be disabled. The following relations are fulfilled: $\Sigma = \Sigma_c \cup \Sigma_{nc}$ and $\Sigma_c \cap \Sigma_{nc} = \emptyset$.

2.2 System behavior

In order to restrict the behavior of the plant G, one or more specifications can be defined, which are usually modeled by a DFSA: $H = (Y, \Sigma_H, f_H, \Gamma_H, y_0, Y_m)$.

A specification limits the behavior of the plant by means of one of the two composition operations defined by the SCT. Informally, it can be said that a composition operation allows two DFSA to run synchronously, which means that certain events will only be created if both DFSA are capable of doing so. In this work, we will consider (synchronous) parallel composition of G and H, defined as:

$$G \| H = Ac(X \times Y, \Sigma_G \cup \Sigma_H, f_{G\|H}, \Gamma_{G\|H}, (x_0, y_0), X_m \times Y_m),$$

where

$$f_{G\|H}((x, y), e) = \begin{cases} (f_G(x, e), f_H(y, e)) & \text{if } e \in \Gamma_G(x) \cap \Gamma_H(y) \\ (f_G(x, e), y) & \text{if } e \in \Gamma_G(x) \text{ and } e \notin \Sigma_H \\ (x, f_H(y, e)) & \text{if } e \notin \Sigma_G \text{ and } e \in \Gamma_H(y) \\ \text{not defined} & \text{other case} \end{cases}$$

and

$$\Gamma_{G\|H}(x, y) = [\Gamma_G(x) \cap \Gamma_H(y)] \cup [\Gamma_G(x) - \Sigma_H] \cup [\Gamma_H(y) - \Sigma_G]$$

$Ac()$ denotes the accessible automaton part, excluding, thus, the states that cannot be accessed from x_0 state.

3 Overview of developed framework

Traditionally, when changes in the production process occur, the PLC, which controls the process, needs to be reprogrammed as well. In order to get highly flexible manufacturing systems, in this work, a framework is proposed with a core element: a service repository. In this proposal, the provided services can be invoked dynamically by external entities (i.e., expert system, task planner) based on the production needs. Therefore, if the production process is altered, then it is only necessary to modify the sequence of invocation of the services. In this way, the main advantage is that the software running on the PLC does not need any modification at all, if all the required services have been taken into account. When certain services are invoked, the system can stop working properly. In order to avoid this, our proposal considers supervising the behavior of the system using SCT.

In our framework, if an input event e_i is triggered by an external entity for a certain service, then $e_i \in \Sigma_c$; if an internal event e_i is created by the plant, then $e_i \in \Sigma_{nc}$. As stated in the previous section, once the plant G is modeled, its behavior is restricted by means of one or more specifications $\{H_1, \ldots, H_k\}$ that can

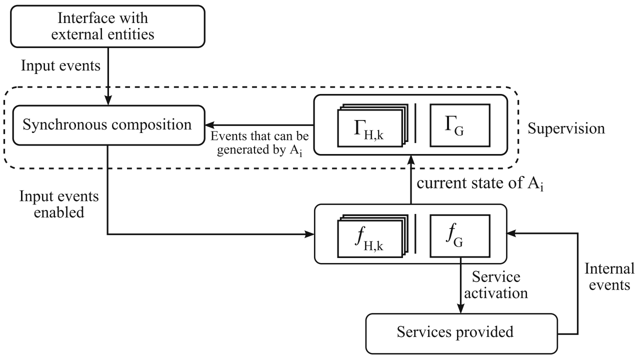

Fig. 1 Global overview of the framework

temporarily disable certain input events. Consequently, no undesired actions will ever take place.

Our framework consists of the following components (Fig. 1):

- *Interface with external entities*. Clients invoke the services that are provided by the system through this interface, and the corresponding input event will be triggered.
- *Supervision*: It is made up of (a) the active event function Γ of A, being $A = \{G, H_1, \ldots, H_k\}$ the set of all the automatons and (b) the synchronous composition $G \| H_1 \| \cdots \| H_k$, where the input events can be disabled if needed.
- *State transition functions*, which evolve in accordance with (1) the input events that have not been disabled previously in the supervision and (2) the internal events of G. In the plant, if an input event causes a transition f_G, it will activate the execution of the corresponding service.
- *Provided services* that gather a specified functionality. Due to design criteria, they are isolated from the supervision stage, so it is possible to perform modifications on the functionality regarding supervision consideration.

In the following, the main behavior will be described. When an input event e_i is triggered, it is processed by the supervisor. The composition operation, taking into account $\Gamma_A = \left\{\Gamma_G, \Gamma_{H_1}, \ldots, \Gamma_{H_k}\right\}$ for each DFSA, decides if e_i must be disabled or not. If e_i is enabled, it is then processed by the f_A of each DFSA, including G. Therefore, if a transition is defined in f_G for the current state of G and e_i appears, the corresponding service will be run. Otherwise, if e_i is disabled, the transition does not happen, and the requested service will never take place. As a result, we make sure that the system behaves according to its specifications at all times.

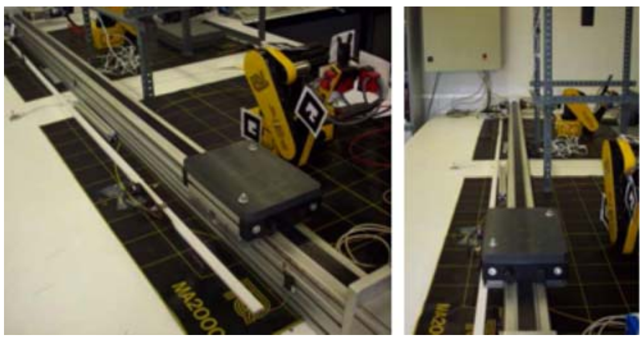
Fig. 2 Considered plant at the case of study

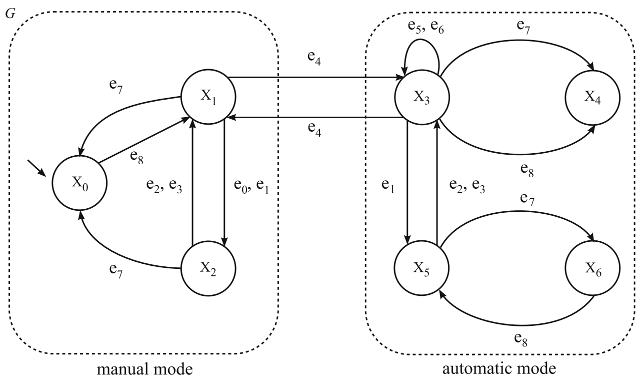

Fig. 3 Plant operation modeled by a DFSA G

4 A case of study

In what follows, we discuss how to use our proposed framework in a typical element of a FMS cell: a conveyor belt. It has been chosen because it is a component found in every cell instance and because the operation specifications are changed frequently, given that the conveyed elements must be moved to different target positions using different velocities as well. The plant (Fig. 2) is made up of a BOSCH conveyor belt with encoder, dynamo, and position sensors. The control is carried out by the Allen Bradley SLC 500 series with the IMC 110 motion control module.

Depending on the operation mode of the IMC, the conveyor belt can be moved manually or automatically. Movements in manual mode are used for initialization and maintenance by an operator and are always executed at constant velocity. Movements in automatic mode are performed when some of the MML programs loaded in the IMC 110 memory are running, and it is possible to modify both the velocity and the acceleration of the motion. In order to work in automatic mode and know where the tray is located, a home operation needs to be performed. For security reasons, emergency stops must be considered when in automatic mode.

Table 1 Defined states in plant G

State	Mode	Conveyor belt state	e-stop
x_0	Manual	Stopped	Yes
x_1	Manual	Stopped	No
x_2	Manual	Moving	No
x_3	Automatic	Stopped	No
x_4	Automatic	Moving	Yes
x_5	Automatic	Moving	No
x_6	Automatic	Moving	Yes

Table 2 Defined events in *G*

Event	Description	Type
e_0	Request for performing a home operation	C
e_1	Request for moving until the target position is reached	C
e_2	Movement completed successfully	NC
e_3	Request for stopping the current movement	C
e_4	Request for changing the IMC 110 operation mode	C
e_5	Request for setting the velocity of automatic movements	C
e_6	Request for setting the acceleration of automatic movements	C
e_7	Emergency stop occurs	NC
e_8	Emergency stop situation is finished	NC

C controllable, *NC* non-controllable

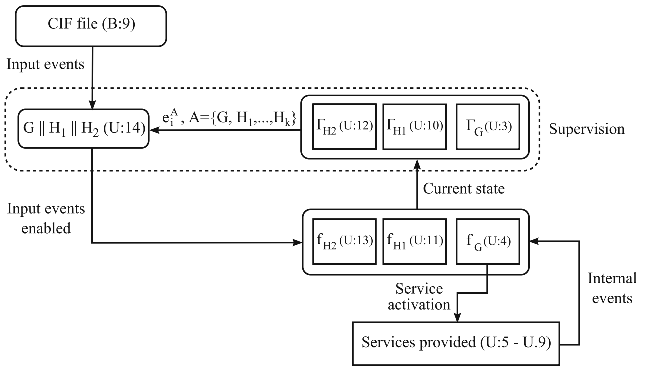

Fig. 5 Implemented supervisor system

4.1 Modelling the plant and the specification

The DFSA G (Fig. 3) models both the manual and automatic modes of the conveyor belt. This plant model DFSA is given as the six-tuple with x_0 as initial state. In the definition of the state set $X_G = \{x_0, \cdots, x_6\}$ (Table 1), three factors have been considered: IMC operation mode (automatic or manual), conveyor state (moving or not), and the presence of an emergency stop. So, in this case, $X_G = X_{G_m}$. Respect to event set Σ_G (Table 2), the set $\Sigma_{NC} = \{e_2, e_7, e_8\}$ is generated from the information collected by the IMC from sensors, $\Sigma_C = \{e_0, e_1, e_3, e_4, e_5, e_6\}$ corresponds to the services that are provided by the system.

The behavior of the plant is controlled by two restrictions:

- R1. The movement to a target position (e_1 event) is not allowed prior to completing a successful home operation.
- R2. The change to automatic mode (e_4 event) is not allowed prior to completing a successful home operation.

Let H_1 and H_2 be the two DFSA that model the restrictions R1 and R2, respectively. According to the H_1 state transition diagram (Fig. 4), the state y_1 represents that the action *performing home operation* is being executed. If it is completed successfully (e_2 event), the state changes to y_2. Given that $e_1 \in \Gamma_{H_1}(y_2)$, then it is possible to perform any type of movement to any position.

Taking into account the state transition diagram (Fig. 5) for H_2, it is only possible to change to automatic mode (e_4 event) if a home operation has been completed successfully. Diagrams H_1 and H_2 are structurally identical, and both restrictions over G could have been modeled by only one specification H. However, modeling several modular specifications H_i, structurally simpler than H, is easier to verify and understand [4].

4.2 Implementation of plant and supervisor systems

In order to implement the supervisor system (Fig. 5), the proposed framework has been followed, taking into account the DFSA G, H_1, and H_2 defined previously. The code has been organized in subroutines stored in PLC files, as indicated in Table 3. The code for the plant and the specifications uses the instruction set of processor SLC 5/03 [2].

For implementing the interface with external entities, we have used the common interface file (CIF) data file, given that is widely used by PLC, including the SLC 5/03. Using a specific protocol (DF1 in Allen Bradley), a client can request a service by activating certain bits in the CIF file.

In the coding of Fig. 6, each input event e_i corresponds to a bit from the CIF processor file. For each

Fig. 4 State transition diagram for H_1 (*left*) and H_2 (*right*)

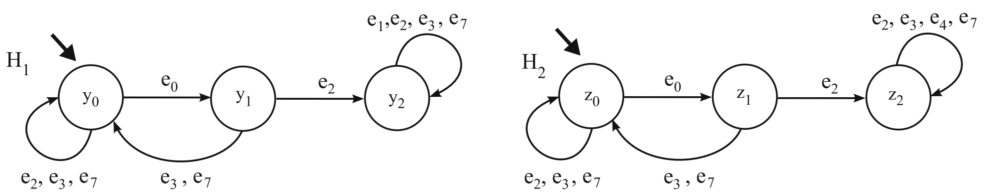

Table 3 Program main files

Id.	Description
U:2	Beginning of the program
U:3	Active event function G
U:4	State transition function G
U:5	Home operation IMC
U:6	Movement to a position
U:7	Change operation mode IMC
U:8	Set the conveyor velocity
U:9	Set acceleration
U:10	Active event function H_1
U:11	State transition function H_1
U:12	Active event function H_2
U:13	State transition function H_2
U:14	Composition $G\|H_1\|H_2$

e_i, the corresponding service bit s_i is defined. When this bit becomes 1, the file (U:5 to U:9) that contains the implementation of this service is executed. The input event e_i is not used because it is only active for one scan cycle, whereas a service usually takes several scan cycles to complete, and therefore, the file that implements it must be called numerous times. In the coding of Γ_G (Fig. 7), non-controllable events (e_2, e_7, and e_8) are ignored. For each event $e_i \in \Sigma_G \cap \Sigma_C$, a bit $e_i{}^G$ is defined, which determines in each program scan if the above event is enabled or not.

With respect to the specifications, given that H_1 and H_2 are structurally identical, only the implementation of H_1 is described. Just as we did with the plant G, in the coding of Γ_{H_1} (Fig. 8), a bit $e_i{}^{H_1}$ is defined for each event $e_i \in \Sigma_{H_1} \cap \Sigma_C$. Concerning the implementation of f_{H_1} in Fig. 9, it can be observed that those transitions that end in the same state ($f_{H_1}(y_j, e_i) = y_j$) are not coded because it does not involve any kind of change in the state of the program.

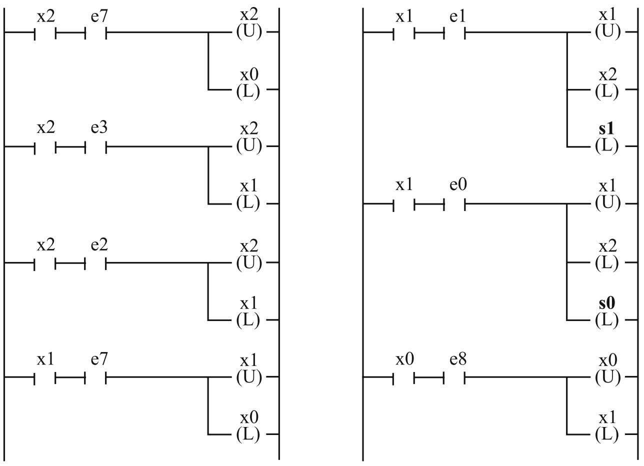

Fig. 6 Fragment of f_G implementation (U:4 file)

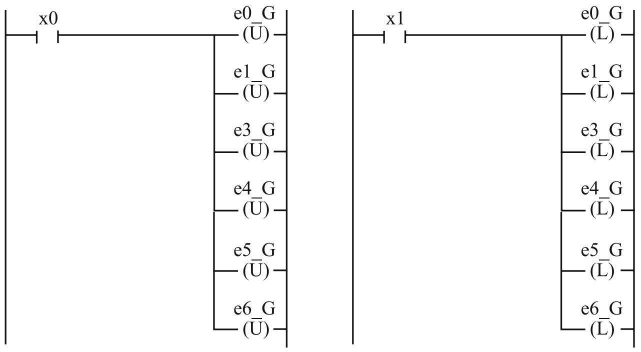
x0 e0_G (U) e1_G (U) e3_G (U) e4_G (U) e5_G (U) e6_G (U)
x1 e0_G (L) e1_G (L) e3_G (L) e4_G (L) e5_G (L) e6_G (L)

Fig. 7 Fragment of Γ_G implementation (U:3)

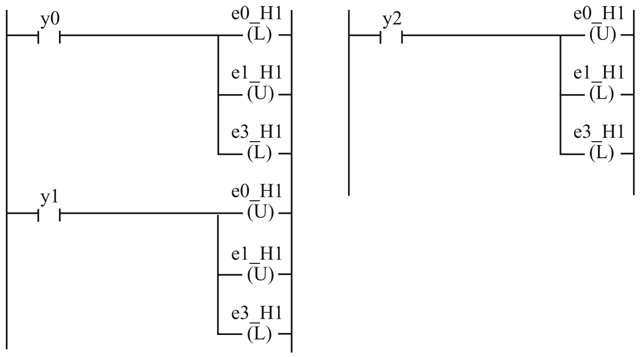
y0 e0_H1 (L) e1_H1 (U) e3_H1 (L)
y1 e0_H1 (U) e1_H1 (U) e3_H1 (L)
y2 e0_H1 (U) e1_H1 (L) e3_H1 (L)

Fig. 8 Γ_{H_1} implementation (U:10)

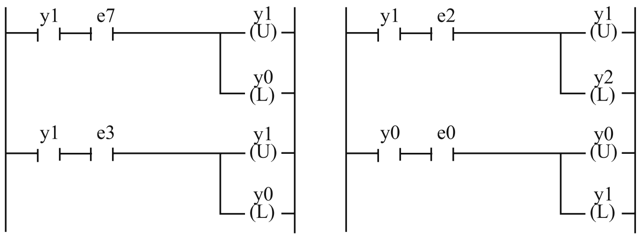

Fig. 9 f_{H_1} implementation (U:11)

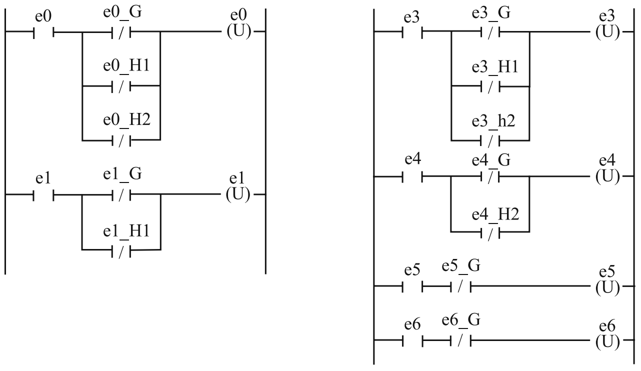

Fig. 10 $G\|H_1\|H_2$ implementation (U:14)

Finally, the file U:14 contains the coding of the synchronous composition operation (Fig. 10) between the plant and the specifications. It is important to stand out that $e_4 \notin \Sigma_{H_1}$ and $e_1 \notin \Sigma_{H_2}$.

5 Conclusions

In this paper, a software framework where the main aim is to make the implementation of a supervisory control system over a PLC easier is proposed. With this framework, the idea is emphasized that the functionalities must be offered with no effects on the robustness of the system operation. The system can be viewed as a well-defined set of services that are requested depending on the production needs. Changes in the production process can affect the order in which the services are requested but not their implementation if they are well defined. However, the possibility that certain event sequences can put at risk the good working order of the system must be considered. In order to avoid these possible incidents, we use the supervised control theory. The proposed framework allows modeling one or more local specifications, which guarantee that the system behaves properly at all times. This is achieved by temporarily disabling the input events which could put the system at risk. Our framework also differentiates between the specifications and the provided functionality. Thus, it is possible to modify both parts independently.

References

1. Beck J, Reagin M, Sweeny T, Anderson R, Garner T (2000) Applying a component-based software architecture to robotic workcell applications. IEEE Trans Robot Autom 16:207–217
2. Bradley A (1996) SLC 500 instruction set reference manual. Allen Bradley Literature
3. Buyurgan N, Saygin C (2006) An integrated control framework for flexible manufacturing systems. Int J Adv Manuf Technol 27:1248–1259
4. Cassandrass CG, Lafortune S (2007) Introduction to discrete event systems, 2nd edn. Springer, New York
5. Chandra V, Mohanty SR, Kumar R (2000) Automated control synthesis for an assembly line using discrete event system control theory. IEEE Trans Robot Autom 14:125–136
6. Curto B, Garcia FJ, Moreno V, Gonzalez J, Moreno A (2001) An experience of a CORBA based architecture for computer integrated manufacturing. In: Proceedings of ETFA-2001. 8th IEEE international conference on emerging technologies and factory automation, vol 1, pp 765–769
7. Hasdemir T, Kurtulan S, Goren L (2008) An implementation methodology for supervisory control theory. Int J Adv Manuf Technol 36:373–385
8. Music G, Matko D (2002) Discrete event control theory applied to PLC programming. AUTOMATIKA 43(1–2)
9. Phoha VV, Nadgar A, Ray A, Phoha S (2004) Supervisory control of software systems. IEEE Trans Comput 53(9)
10. Ramadge PJ, Wonham WM (1987) Supervisory control of a class of discrete event processes. SIAM J Control Optim 25(1):206–230