Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

Hierarchical control architectures in industrial automation: a design approach based on the generalized actuator concept

Eugenio Faldella [*] **Andrea Paoli** [**,1] **Matteo Sartini** [**]
Andrea Tilli [**]

[*] *DEIS - Department of Electronic, Computer Science and Systems,*
University of Bologna,
Viale Risorgimento, 2 - 40136 Bologna, ITALY,
efaldella@deis.unibo.it
[**] *Center for Research on Complex Automated Systems (CASY)*
Giuseppe Evangelisti, DEIS - Department of Electronic, Computer
Science and Systems, University of Bologna,
Viale Pepoli, 3/2 - 40123 Bologna, ITALY,
{andrea.paoli ; matteo.sartini ; andrea.tilli}@unibo.it

Abstract: In this paper an effective design approach to the design of hierarchical control architectures for the automation industrial plants is presented. The main characteristic of the solution is the clear and structural separation between “policies” and “actions” deriving from the use of a novel abstract entity in modelling automation plants: the Generalized Actuator. Particular attention is paid to illustrate how to define generalized actuators starting from a “bare plant”. The potentialities of this method are emphasized by means of a case study.

Keywords: Modelling and design of automation control systems; Process supervision; Flexible and reconfigurable automation systems

1. INTRODUCTION

In software engineering, concepts as modularity, encapsulation, composability and reusability are strongly emphasized and profitably realized in the so-called object-oriented methodologies. These methodologies are fruitfully pervading the industrial automation world too, as testified not only by current availability of commercial products conforming, at least to a certain extent, to the standards defined for this specific domain by International Organisms, such as IEC and OMG, but also by some interesting proposal about generally applicable modelling and design frameworks recently published in the scientific and technical literature. From the latter point of view, in (Vyatkin and Hanisch [2001]) the framework of IEC61499 is exploited to define a formal modelling suitable for verification. In (Bonfè and Fantuzzi [2004]) the *Mechatronic Object* has been introduced to deal with mechanical and electronic issues involved in the automation of industrial plant. This approach has been further extended in (Bonfè et al. [2006] where a solid unification of dynamic systems and industrial control software modelling is proposed. In (Thramboulidis [2005]) a model integrated paradigm is introduced to represent mechatronic systems. Differently, in (Ferrarini et al. [2006]) the *Control Module* is introduced following the agent paradigm to achieve a modular representation of automation functions for flexible manufacturing systems.

As a matter of fact, a key element for the effectiveness of a proposed modelling framework is the correlation with a clear procedure to deal with it. Taking inspiration from this basic consideration, the main focus of this paper is to present a *modelling framework* and a *design procedure* to realize automation functions exploiting a clear and structural separation between *Policies* and *Actions*. Toward this purpose, a novel entity is introduced for modeling industrial automation systems: *the Generalized Actuator* (GA). The main characteristics of the proposed modelling framework and design procedure are the following:

- Introduce a straightforward way to encapsulate “actuation mechanisms”, using GA;
- Effectively support hardware virtualization, component interoperability and reusability;
- Allow hierarchical management of a plant, separating control policies from actuation mechanisms;
- Allow detection of anomalous situations following a distributed hierarchical approach.

The paper is organized as follows. In Section 2 a case study is presented to emphasize the basic idea of the proposed methodology. It’s Generalization and formalization is presented in Section 3. The potentialities of the approach, concluding remark and future work directions are outlined in Section 4.

2. CASE STUDY

To introduce the design procedure, let us start presenting a simple example: a marking machine that marks some packs in a production line. We will deal with the different

[1] Corresponding author.

978-1-1234-7890-2/08/$20.00 © 2008 IFAC 10.3182/20080706-5-KR-1001.3528

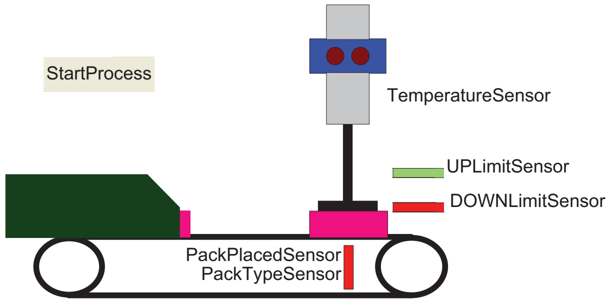


Fig. 1. The considered production line.

control functionalities desired for the system in different steps; namely these functionalities are:

(1) Mark a pack using pack presence sensors;
(2) Control the stamping tool temperature;
(3) Control the tool temperature according to a suitable policy.

The system to control is schematically shows in Figure 1; it is composed by a conveyor belt that feeds some packs produced by an upstream machine, and a stamping machine that marks the packs. The conveyor belt is actuated through the command signal `BeltMotorON`; the system is equipped with a sensor that indicates when a pack is in the marking position (signal `PackPlacedSensor`). The hydraulic ram that actuates the mark has two possible directions of movement, decidable through command signals `RamMotorON` and `RamMotorDirection`, namely upward direction command can be issued through the combination `RamMotorON`=1 and `RamMotorDirection`=1, while the downward direction is imposed through the combination `RamMotorON`=1 and `RamMotorDirection`=0. Two sensors indicate the up limit stop (signal `UPLimitSensor`), and the down limit stop (signal `DOWNLimitSensor`) of the hydraulic ram. When a new pack arrives under the ram, this must reach its downward position and stop for 0.2 seconds; after this time interval the ram must reach its upward limit while the pack is expelled. During the marking interval the tool temperature has to be maintained at a specified level, depending on a policy defined later, using an heating system activated by `HeatingON` and an analog temperature sensor `TemperatureSensor`. The overall process must start when the command `StartProcess` is active and should stop when `StartProcess` becomes false. In Table 1 a description of signals used in the example is given.

2.1 A classic design procedure

First of all we present a “common” solution for the considered problem, under the following hypotheses.

- Suppose the pack presence sensor to be ideal considering that the signal `PackPlacedSensor` immediately raise when a pack is in the correct marking position.
- Assume a policy requiring a constant temperature, `TemperatureReference`, for the marking tool.

In Figure 2 the “common” SFC solution is reported and it reflects the usual approach adopted in industrial automation design. On the left hand side, a sequence to handle the conveyor belt and mark ram coordinate motions is depicted, while on the right hand side a tool temperature controller, based on hysteresis mechanism is presented. In this scheme the functioning logic behind the overall process is hidden in the graph and it is impossible to distinguish between the implementation of the main functions of the systems. As a matter of fact, despite the use of SFC language, the designed solution lacks of separation between logic policies and actuation mechanisms, reflecting into a lack of reusability and modularity, as it can be noted considering the following modifications to the plant and policies.

(1) Suppose that the considered system is equipped with a presence sensor which cannot be considered as ideal: signal `PackPlacedSensor` becomes true as soon as the pack reaches the sensor, but this position is not centered below the ram; for this reason, once sensor is covered by the pack, the belt should continue moving for a given time that depends on its actual speed in order for the pack to reach the correct position.
(2) Suppose that the system can manage two different products and, depending on the kind of the pack (distinguished through a sensor readings `PackTypeSensor`), the marking action should be performed at different temperatures (i.e. the reference for the temperature control changes according to the actual product).

The control logic for this more involved situation is depicted in Figure 3; the reader can observe that the new solution is slightly different from the starting one presented in Figure 2. But it is even more interesting noting that the modification (1) is related to an action sensor (in general to an action mechanism), while the modification (2) is related to a policy change, but this characteristics are not clearly distinguishable in the proposed solution (see Figure 3, with the highlighted modifications). This fact clearly testifies the mixing of mechanisms and policies which minimize modularity and reusability properties.

Another worthy remark concerning the solutions of Figures 2 and 3 is to note that the diagnostic phase has been completely disregarded. Even diagnosis of anomalous situation can be considered at two different levels: detection of mechanisms failures (e.g., sensor or actuator faults, components malfunctioning, etc.) and functional anomalies (e.g., forbidden control sequences that occur due to external influences). Having a control logic as the one presented in Figure 3 prevents from obtaining a separation between mechanisms diagnostics and policies diagnostics; this aspect is fundamental when we require the system to be tolerant to faults having to distinguish between mechanisms reliability (e.g. considering redundancy) and policies reconfigurability (e.g. switching between different control logic).

Taking motivation from the enlightened drawbacks, a novel approach procedure is presented in next section. The procedure is devoted to obtain a reusable, modular control logic by exploiting a structural separation between mechanisms and policies.

Signal	Meaning	Type
`StartProcess`	Operator command to start the process.	DIGITAL
`BeltMotorON`	Command signal to move the conveyor belt.	DIGITAL
`PackPlacedSensor`	Pack presence sensor readings.	DIGITAL
`PackTypeSensor`	Pack type sensor readings.	DIGITAL
`RamMotorON`	Command signal to move the hydraulic ram.	DIGITAL
`RamMotorDirection`	Command signal to motion direction for the hydraulic ram.	DIGITAL
`UPLimitSensor`	Hydraulic ram upward limit switch signal.	DIGITAL
`DOWNLimitSensor`	Hydraulic ram downward limit switch signal.	DIGITAL
`TemperatureSensor`	Tool temperature sensor readings.	ANALOG
`HeatingON`	Command to warm up the marking tool.	DIGITAL

Table 1. List of signals used in the benchmark example.

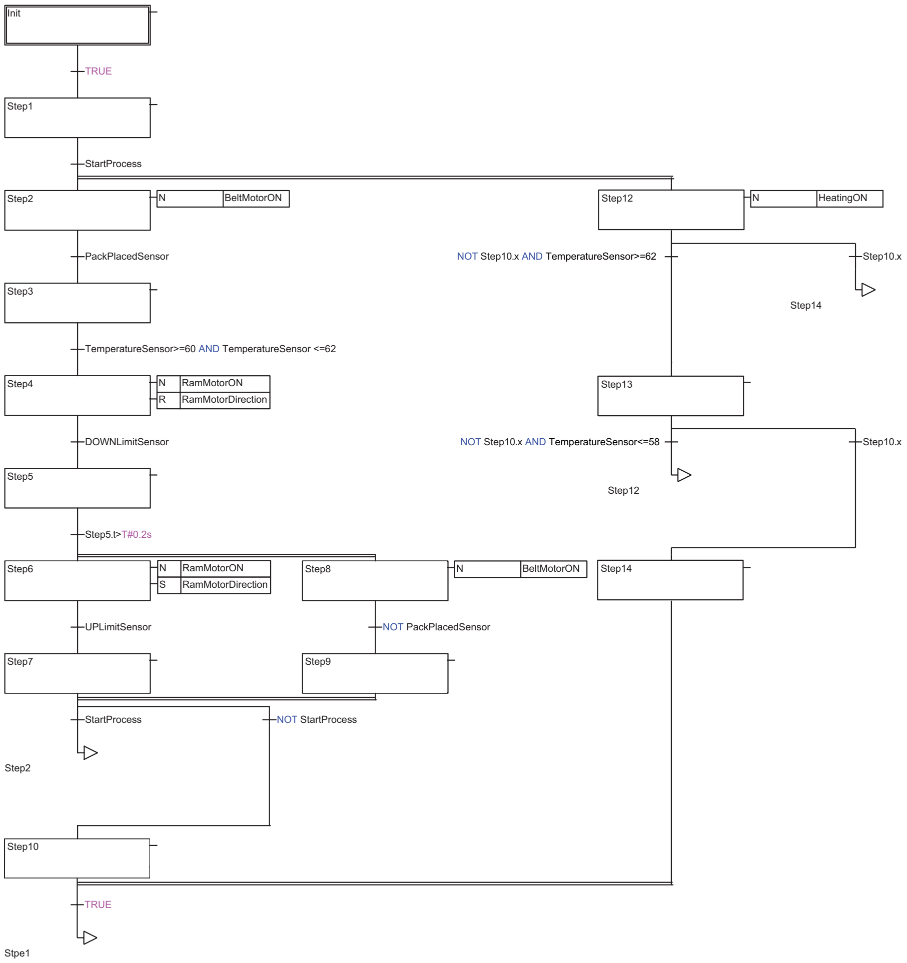


Fig. 2. “Common” SFC solution for the case study.

2.2 A solution to the benchmark example based on the generalized actuator concept

The novel approach proposed in this Section starts from the idea of considering logic control as a recipe mainly composed by two ingredients:

- a set of basic actions;
- one or more desired sequences to coordinate actions execution.

The first ingredient represents mechanisms of functionality implementation, while the second represents the control policy. As enlightened previously the needs for reusable, modular control software require the two to be completely independent. First of all, all the action/mechanisms are defined, using the GA entity; afterwards the overall control policy is considered.

The first step for mechanism definition is to identify basic actions that cannot be reasonably furthermore decomposed. For the considered system the basic actions to perform are (1) move the pack in marking position, indicated as `Positioning`; (2) expel the marked pack, indicated as `Expulsion`; (3) move the ram upward, indicated as `RamGoUp`; (4) move the ram downward, indicated as `RamGoDown`; (5) control the heating system to maintain a given temperature on the marking tool, indicated as `TemperatureControl`.
Each basic action is then associated with a set of actuators and sensors that physically perform the action. As

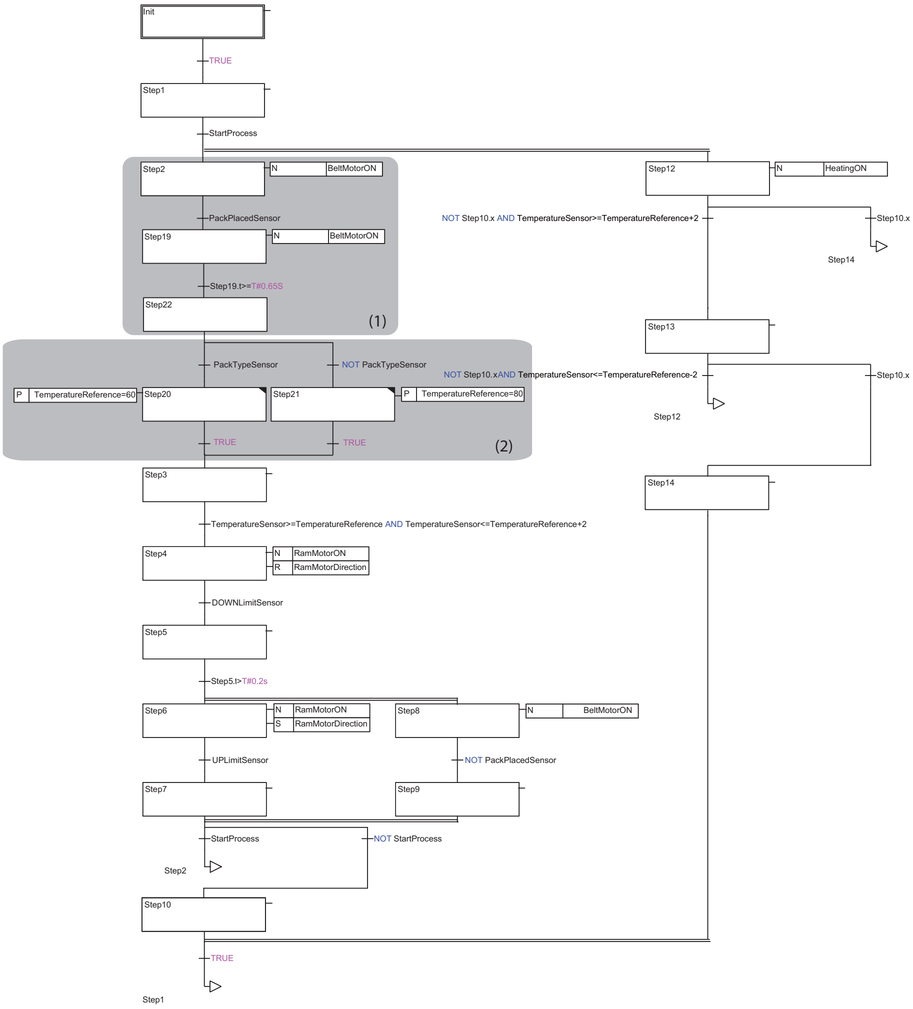


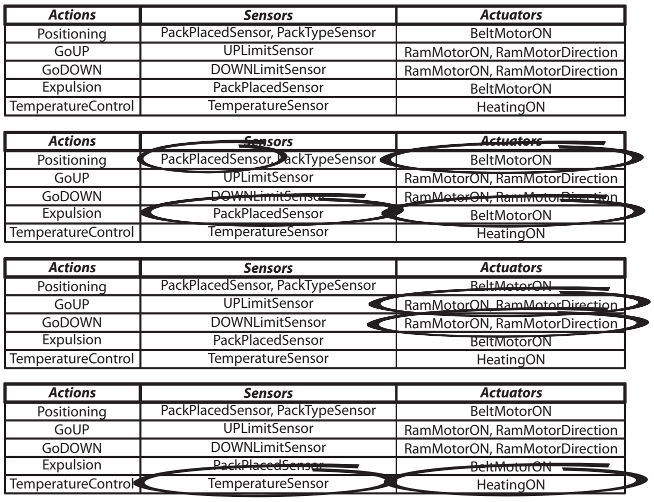
Fig. 3. Solution of Figure 2 adapted to plant and policies modifications.

depicted in Figure 4, action Positioning and `Expulsion` both involve sensor `PackPlacedSensor` and actuator `BeltMotorON`. Actions `RamGoUp` and `RamGoDown` involve sensors `UPLimitSensor` and `DOWNLimitSensor`, respectively, and actuators `RamMotorON` and `RamMotorDirection`, both of them. Finally action `TemperatureControl` involves sensor `TemperatureSensor` and actuator `HeatingON`. At this point, the proposed subsequent step is the effective definition of GAs following this basic concept (which is a sort of definition of a GA): every GA is a "virtual" actuator with the following characteristics:

- it is in charge of the execution of a small subset of the basic actions identified in previous steps (hence it handles actuators and sensor associated to them);
- it is *always "alive"* during the operations of the automation plant, even if no specific action is required to it.

In order to give effective "guide-lines" for this phase, the following rules are introduced:

- the union of the actuators associated with the set of GAs must be equal to the whole actuators set of the system (for what concern the sensors usually the same condition should be satisfied but it is not mandatory);



Actions	***Sensors***	***Actuators***
Positioning	PackPlacedSensor, PackTypeSensor	BeltMotorON
GoUP	UPLimitSensor	RamMotorON, RamMotorDirection
GoDOWN	DOWNLimitSensor	RamMotorON, RamMotorDirection
Expulsion	PackPlacedSensor	BeltMotorON
TemperatureControl	TemperatureSensor	HeatingON

Actions	~~***Sensors***~~	~~***Actuators***~~
Positioning	PackPlacedSensor, PackTypeSensor	BeltMotorON
GoUP	UPLimitSensor	RamMotorON, RamMotorDirection
GoDOWN	~~DOWNLimitSensor~~	~~RamMotorON, RamMotorDirection~~
Expulsion	PackPlacedSensor	BeltMotorON
TemperatureControl	TemperatureSensor	HeatingON

Actions	***Sensors***	***Actuators***
Positioning	PackPlacedSensor, PackTypeSensor	~~BeltMotorON~~
GoUP	UPLimitSensor	RamMotorON, RamMotorDirection
GoDOWN	DOWNLimitSensor	RamMotorON, RamMotorDirection
Expulsion	PackPlacedSensor	~~BeltMotorON~~
TemperatureControl	TemperatureSensor	HeatingON

Actions	***Sensors***	***Actuators***
Positioning	PackPlacedSensor, PackTypeSensor	BeltMotorON
GoUP	UPLimitSensor	RamMotorON, RamMotorDirection
GoDOWN	DOWNLimitSensor	RamMotorON, RamMotorDirection
Expulsion	~~PackPlacedSensor~~	~~BeltMotorON~~
TemperatureControl	TemperatureSensor	HeatingON

Fig. 4. Actions, sensors and actuators of the systems.

- pursuing a non interference idea, sets of sensors and actuators belonging to different GAs must be disjointed.

In the considered example, the situation depicted in lower part of Figure 4 is obtained. Looking for common equipment used in different actions (circled in Figure 4), leads to group them in three GAs respectively devoted to the positioning of packs, marking of packs and temperature control.

From the considered example, it is immediate to note that there exist two different kinds of actions and, consequently, of GAs; there are actions which structurally terminate after a finite time (e.g. action `Positioning` implies moving the belt until the pack reaches the marking position), while there are others which, in principle, could continue for an infinite time and whose termination has to be decided “externally” (e.g. action `TemperatureControl`).

The GAs associated to the first kind of actions are denominated **Do-Done** GA. They are characterized by a input signal `Do` used to command the starting of an action, an input signal `DoWhat` to specify what kind of action has to be performed (if more than one is available) and an output signal `Done` to signal when the action has terminated successfully.

Differently, the GAs associated to the second kind of actions are denominated **Start-Stop** GA. Their characteristic I/O signals are the input `Start` to command the beginning of an action, defined by the input `StartWhat`, and the input command `Stop` to stop the action.

Other I/O can be present for both types of GA, different I/O categories are described in details in next section.

For the presented example, the following GAs can be defined:

- `PackMotion`, a Do-Done GA that is devoted to packs positioning;
- `RamMotion`, a Do-Done GA that is aimed at moving the hydraulic ram;
- `TemperatureControl`, a Start-Stop GA that is aimed at controlling the stamping tool temperature.

These three GAs are described and realized using the Function Block (FB) formalism defined in IEC61131-3 in Figure 5. It is worth noting that the FB are not used by chance, in fact they represent Program Organization Units (POUs) which have to be always active during overall control execution. The input/output interfaces for these GAs `PackMotion`, `RamMotion` and `TemperatureControl` are presented in tabular form in Figure 6. It is worth noting that the internal variables with “Port” suffix are used to identify the physical sensor and actuators linked to the GAs.

For the sake of brevity we describe now only the implementation of the `PackMotion` GA, based on SFC formalism and depicted in Figure 7. After its activation the GA is in `Init` state in which some initialization operations are performed; when these actions are accomplished, the GA moves into the `Ready` state in which waits for the `Do` command. Right after receiving the `Do` signal the GA moves into a `Busy` macro-state; the first action within the macro-state is to distinguish between the `Positioning` action and the `Expulsion` action. In the first case the belt is moved until `SensorPosition` (which is sensor `PackPlacedSensor`) reads the presence of a pack; due to the non ideality of the sensor the belt continues to be actuated for a given time specified in parameter `PositioningDelayTime`. After having positioned the pack under the hydraulic ram the signal coming from sensor `PackTypeSensor` is filtered in state `PackModel` and

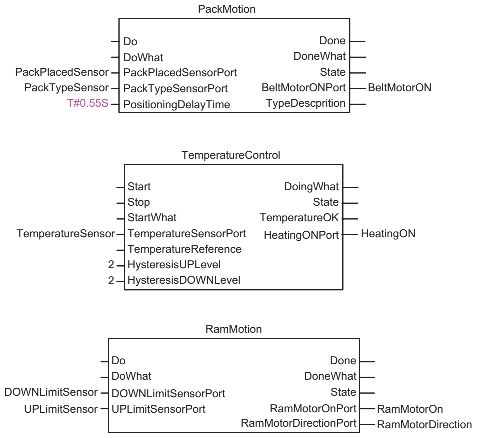


Fig. 5. Function Block representation of GAs `PackMotion`, `RamMotion` and `TemperatureControl`.

GA	IN		OUT	
	PortName	Values	PortName	Values
PACK MOTION	DO	TRUE FALSE	Done	TRUE FALSE
	DOWhat	Positioning Expulsion	DoneWhat	Positioning Expulsion
	PackType SensorPort	PackType Sensor	State	Idle Busy Ready
	Positioning DelayTime	T#0.55S	Type Description	Pack0 Pack1
	PackPlaced SensorPort	PackPlaced Sensor	Belt MotorONPort	Belt MotorON
RAM MOTION	DO	TRUE FALSE	Done	TRUE FALSE
	DOWhat	RamGoUP RamGoDOWN	DoneWhat	GoUP GoDOWN
	DOWNLimit SensorPort	DOWNLimit Sensor	State	Idle Busy Ready
	UPLimit SensorPort	UPLimit Sensor	RamMotor ONPort	RamMotor ON
			RamMotor DirectionPort	RamMotor Direction
TEMPERATURE CONTROL	Start	TRUE FALSE	Done	TRUE FALSE
	Stop	TRUE FALSE	DoingWhat	Warming60 Warming80
	StartWhat	Warm60 Warm80	State	Idle Busy Ready
	Temperature SensorPort	Temperature Sensor	Temperature OK	TRUE FALSE
	Temperature Reference	60 80	HeatingON Port	HeatingON
	Hysteresis UPLevel	2		
	Hysteresis DOWNLevel	2		

Fig. 6. Input/output interfaces for GAs `PackMotion`, `RamMotion` and `TemperatureControl`.

the measured pack type is communicated through signal `PackDescription`. Having positioned the pack and communicated its type the `Done` signal is issued and the GA moves back in state `Ready`. If the action to performed was `Expulsion` the GA just actuates the belt until the presence sensor reads that the pack is away from the marking position.

It is worth noting that the particular strategy required to handle the non-ideality of sensor `SensorPosition` (introduced by modification (1) of Section 2.1, and here highlighted in the shaded box (1) of Figure 7) is now clearly encapsulated inside the `PackMotion` GA, enlightening that

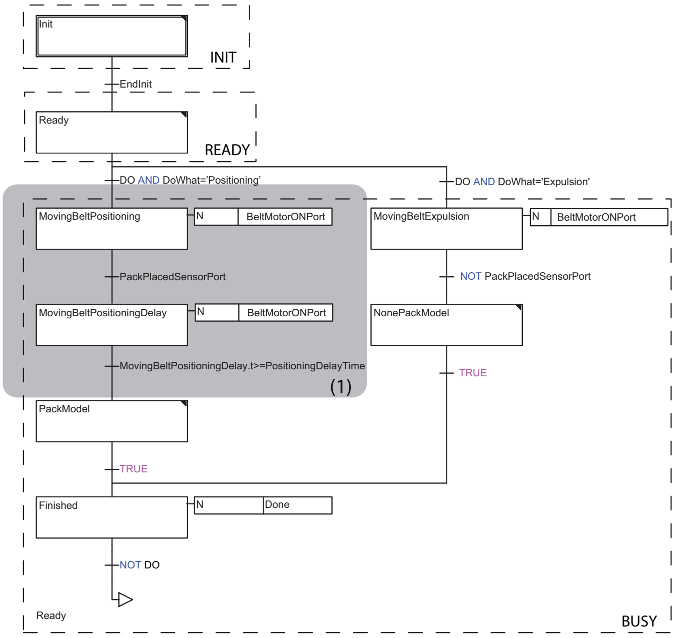


Fig. 7. SFC implementation of the PackMotion GA.

this strategy is simply related to a particular mechanism, it doesn't affect other mechanisms and the overall policy.

Once all the GAs are completely defined, the overall control policy can be designed. The supervision policy, adopted to coordinate the three GAs of the proposed example, is defined using SFC implementation as depicted in Figure 8. After an initialization phase performed in state Init, the system moves in state Ready in which wait for the command StartProcess. After this last is issued from the operator, the logic moves into state StartTemperature control in which the Start-Stop GA TemperatureControl is activated and the reference temperature is set to the lower one using signal TemperatureReference. At this point, entering state Positioning, the PackMotion GA is activated with Doingwhat='Positioning' in order to have a pack positioned in the marking station. When the signal PackMotion.Done is true, the logic is sure that the marking operation can start, therefore, according to the pack type communicated by the PackMotion GA it sets the correct temperature reference and wait for the signal TempCont.TemperatureOK generated by the TemperatureControl GA to state that the desired temperature has been reached. At this point in state MarkDown the RamMotion GA is activated with DoWhat='RamGoDOWN' in order to move downward the hydraulic ram; when the GA communicates that the desired movements has been performed (signal Mark.Done), the logic waits for 0.2 seconds and, after that, in parallel move upward the ram activating the RamMotion GA with DoWhat='RamGoUP' (state MarkUp) and expel the marked pack activating the PackMotion GA with DoWhat='Expulsion' (state Expulsion). When both the actions are finished (signals Mark.Done and Pack.Done both raised to one). Finally if the StartProcess is still active the logic moves back to positioning state to let a new pack to be manipulated, while, if the StartProcess signal is false the TemperatureControl GA is stopped and the SFC moves back to Ready state. It is worth noting that in this case the particular strategy for the temperature reference setting (required by the modification (2) reported in Section 2.1), and highlighted here in the shaded box (2) of Figure 8) is now clearly classified as part of the supervision policy, without any connection to the mechanisms encapsulated in GAs.

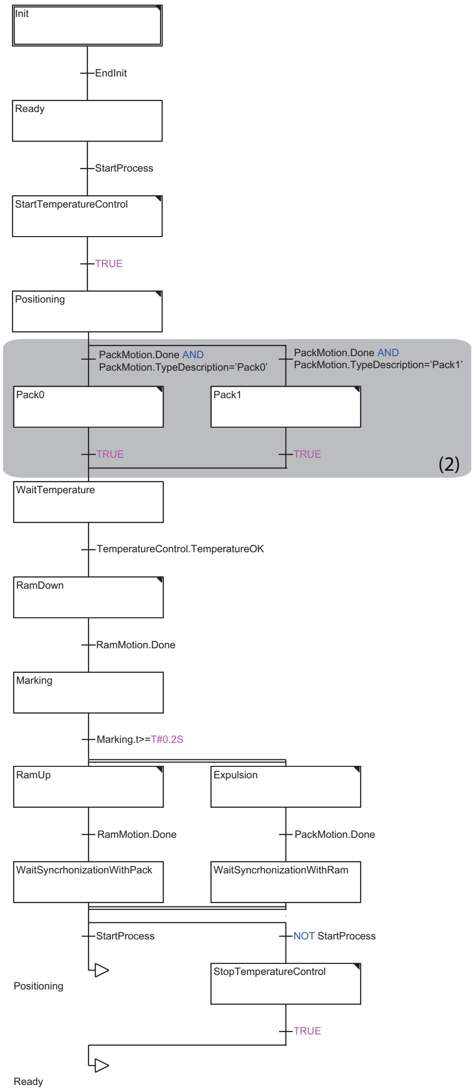


Fig. 8. SFC implementation of the supervision policy for the benchmark example.

To conclude this Section it is possible to note that now policies are completely independent from mechanisms: changing sensors or actuators will not affect the policy but only GAs implementation, being the supervision logic influenced just by signals generated by GAs through physical signals filtering. Implementing this solution instead to the one shown in Figure 3 let the designer to obtain a fully encapsulated and modular software which can be easily modified, enriched with new functionalities and reused. Moreover, even if this point is not stressed in this work, it is immediate to understand that the diagnostics phase becomes easier thanks to the structure of the control logic, being possible to distinguish between mechanisms diagnostics (which can be performed on-the-fly by GAs, even when they are idle) and policy diagnostics performed just considering the command sequences issued at high level and the responses of GAs.

3. GENERALIZED ACTUATOR DEFINITION AND DESIGN PROCEDURE FORMALIZATION

We are now ready to define the structure of a GA specializing its input/output interface and its basic dynamics by means of a state diagram; after that, generalizing the procedure proposed in Section 2.2 to solve the benchmark example, we furnish some guidelines to design the control logic using the GA approach.

In Figure 9 the interface section of both Do-Done GA (Figure 9(a)) and Start-Stop GA (Figure 9(b)) are depicted; in both cases the interface can be mainly divided in two sections in the following described.

(1) **Interface to policy**: this section represents the input/output section between the GA and the supervision policy. It can be further decomposed in two subsections separating the standard communications between the GA and the policy and all the case dependent communications.

Standard interface: embeds all command inputs for the GA and the outputs that communicate the actual state of the GA and the task that it is accomplishing. More in details the Do-Done GA will receive as command the `Do` signal to start operations and the `DoWhat` signal to specify the desired action, while the Start-Stop GA will be commanded through inputs `Start` to start operations and `Stop` to conclude operations, and through signal `StartWhat` to define the required action. In both cases input signals `Alarm`, `AlarmType` can be used to communicate to the GA the occurrence of an external anomalous situation. The outputs of this section are, for the Do-Done GA, the `Done` signal by which the GA communicate that the task has been performed and the `DoneWhat` signal by which the terminated task is specified; the Start-Stop GA outputs are the signal `DoingWhat` representing the task that the GA is performing. In both kind of GAs, a `State` signal communicate the actual state in which the GA is evolving.

Communications: represents all the non standard communications between the policy and the GA, as the results of sensor readings filtering (e.g. the `PackDescription` signals in `PackMotion` GA, that distinguish between two different kind of packs filtering the sensor readings `PackTypeSensor`, see Section 2.2).

(2) **Low level interface**: this section contain all the interfaces with the low level world; even this section can be further decomposed in two sub sections considering the constant parameters used by the GA separated from the physical interconnection with the plant.

Constant parameters: contains all the inputs by which it is possible to give a constant value to characteristic parameters of the GA (e.g. in `PackMotion` GA the input `PositioningDelayTime` by which define the time interval between the activating instant for sensor `PackPlacedSensor` and the instant in which the pack reaches the marking position, see Section 2.2).

Plant I/O link: is the real interface with the plant and contains as inputs all the links to sensors and as outputs the links to actuators. In this way the physical connection between the GA and the plant is completely hidden to the high level control policy.

In Figure 10 the interfaces of the three GAs for the benchmark system are specialized following the classification just described. This I/O structure let to consider for each GA a virtual terminal board so that the implementation mechanisms are hidden to the policy. For example in the `RamMotion` GA the `DoWhat` command can assume the values `RamGoUP` and `RamGoDOWN` reflecting respectively in physical commands (`RamMotorON`=1 and `RamMotorDirection`=1) and (`RamMotorON`=1 and `RamMotorDirection`=0).

The GA should then be designed considering as *Reference Model* the event driven evolution in Figure 9(c). In the depicted automaton it is possible to distinguish the states in the following described.

`Init`: this state is the initial one and becomes active as soon as the GA is activated (usually at the beginning of operations). It represents the state in which initialization actions are performed; the GA moves out from this state when a signal `EndInit` communicates that the initialization operations are concluded forcing the GA to move in `Ready` state.

`Ready`: in this state the GA is ready to perform the desired operation and is waiting for the `Do` or `Start` command to move to `Busy` state.

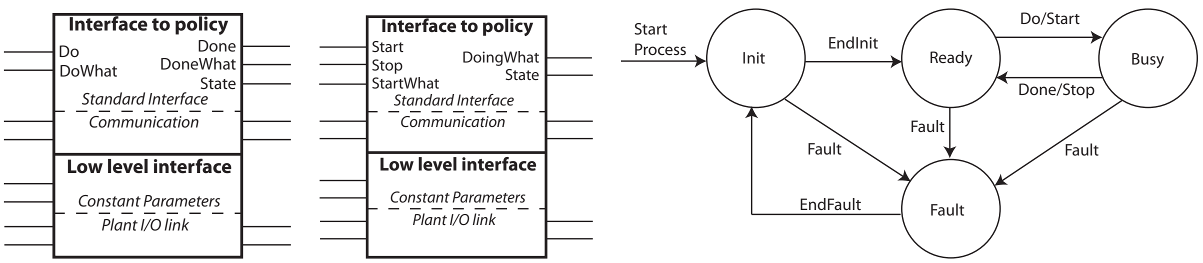
`Busy`: after the command issued by the policy the GA starts performing its required task communicating with the high level policy information on the accomplishment of the function (e.g. information on the quality of the operations). The GA remains into this state until the task is finished and the signal `Done` is raised (Do–Done GA) or until the `Stop` signal (Start–Stop GA) is issued by the policy. In these cases the GA moves back to state `Ready`.

`Fault`: from any state a signal `Fault` (used to communicate some anomalies) can force the GA to move into a `Fault` state in which some counteractions are taken. Note that the `Fault` signal can be both due to external commands (e.g. an alarm issued by an external operator), to internal diagnostics or to wrong logic operations. When the alarm situation is concluded (signal `EndFault`) the GA returns in the `Init` state to be reinitialized.

It is important to stress that each of the states just described represent a set of states; in this sense the automaton in Figure 9(c) plays the role of a logic design pattern similarly to GEMMA diagram (see Moreno and Peulot [2002]). Going back to Figure 7 the SFC state is clustered according to the general reference model; note that fault communications and state are not present, since this topic has not been considered in the example for the sake of brevity.

To conclude this Section we briefly summarize the design procedure based on GAs and introduced in Section 2.2:

(1) Identify basic actions of the process;
(2) Define Do-Done actions;
(3) Define Start-Stop actions;



(a) Do-Done GA interface. (b) Start-Stop GA interface. (c) State diagram for the GA.

Fig. 9. Interfaces of GAs.

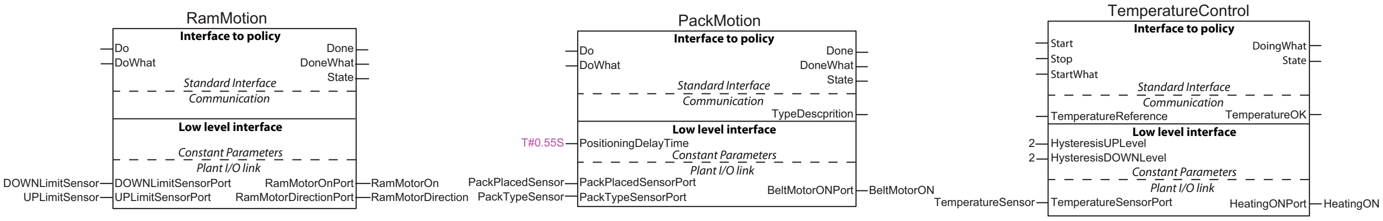


Fig. 10. Interface classification for the `PackMotion`, `RamMotion` and `TemperatureControl` GAs.

(4) Identify the GAs by grouping actions with overlapping sets of sensors or actuators;
(5) Design each GA by:
    - Defining its interfaces;
    - Designing the actuation logics according to reference model in Figure 9(c);
    - Designing the internal diagnostics and quality assessment procedures (not considered in this work);
(6) Design the high–level policies

4. CONCLUSIONS

In this work an effective design procedure for industrial automation systems has been described. This procedure exploits the structure of a novel abstract element, called Generalized Actuator, to guarantee hierarchical management of the plant, separating control policies from actuation mechanisms.

Actually the emphases has been put on the improvements obtained in terms of modularity and reusability, but it is also important to stress that the proposed procedure, that reflects into a hierarchical architecture, is the starting step towards a hierarchical diagnostics systems, in which mechanisms fault diagnosis is separated from policy safety verification, and especially towards a hierarchical reconfiguration system in which fault counteractions can be taken separately at mechanisms level and/or at policy level.

The presented results have been developed to obtain a product that is norm IEC61131–3 compliant, moreover, as enlightened in the paper, these results are fully integrable with those regarding other object-oriented approaches to industrial automation as those in Section 1.

Finally the further developments will regard the following points:

- including hierarchical diagnostics systems design steps in the proposed pattern;
- defining quality/safety/tolerance indexes to be taken into account at different level of the architecture, designing different reconfiguration strategies;
- move towards formalisms defined in IEC61499.

ACKNOWLEDGMENTS

The authors wish to thank Prof. Claudio Bonivento of the University of Bologna and Dr. Pierantonio Ragazzini of IMA (Industria Macchine Automatiche) of Bologna for several fruitful discussions. The authors gratefully acknowledge also the contribution of IMA in developing a prototype test bed and making the lab experiments.

REFERENCES

M. Bonfè and C. Fantuzzi. Application of object-oriented modeling tools to design the logic control system of a packaging machine. *IEEE International conference on Control application Industrial Informatics*, pages 506 – 574, 2004.

M. Bonfè, C. Fantuzzi, and C. Secchi. Behavioural inheritance in object-oriented models for mechatronic systems. *International Journal of Manufacturing Research*, 1(4):421 – 441, 2006.

L. Ferrarini, C. Veber, and V. Schirò. A modular modelling and implementation of automation functions for flexible manufacturing systems. *ANIPLA international Congress 50 Anniversary 1956-2006, Methodologies for Emerging Technologies in Auotomation*, 2006.

S. Moreno and E. Peulot. *Le GEMMA: modes de marches et d'arrest, Grafcet de coordination des taches, Conceptions des Systemes Automatises de Production surs.* Editions Casteillal, 2002.

K. C. Thramboulidis. Model integrated mechatronics-towards a new paradigm in the development of manufacturing systems. *IEEE Transaction on Industrial Informatics*, 1:54 – 61, 2005.

V. Vyatkin and H. M. Hanisch. Formal modeling and verification in the software engineering framework of iec 61499: a way to self-verifying systems. *IEEE International conference on Emerging Technologies and Factory Automation*, 2(113–118), 2001.