The Building of Distributed Automation Control Systems based on PLC Programming and Extends IEC 61131 Standard

Zhou Baiqing

Zhejiang Tongji Vocational College of Science and Technology, Zhejiang China 311231
zjzbq1976@yahoo.cn

Abstract

Today industrial automation control requirements include capability to implement application involving widely distributed devices, high reuse of software components, formal verification that specifications are fulfilled. In this paper an object oriented tool which extends 61131 standard with object oriented elements, the programming language SFC together with a proper way to organize the inputs and outputs of FBs and supervisory control are proposed to implement industrial automation control systems based on a service oriented architecture to meet the new challenges of this field.

Keywords: Distributed Automation Control Systems, PLC, IEC 61131

1. Introduction

Industrial controllers, e.g. PLCs, are today largely based on IEC 61131 [8]. The main reason for the development of IEC 61131 was the existence of a huge variety of languages for PLC programming and therefore an unsatisfactory situation for the users. Most PLC vendors are compliant to the definitions of the IEC 61131 standard and similar means for programming are available on the different PLC platforms. Moreover, huge amount of manpower and money has been spent in industry in order to establish applications and libraries for IEC 61131 and this makes difficult to adopt new standards or new paradigms, at least in the next future, unless certain compatibility with respect to 61131 is preserved [18].

However, the control of widely distributed systems requires more powerful tools. Here, we want point out on two main innovations in this context.

1. Object oriented extension. An IEC 61131 maintenance group is discussing a working draft of the next edition. This draft contains a proposal for object-oriented extensions to be added to the IEC 61131-3. The IEC 61131 already contains a simple class concept, the function block. A function block has an internal state, a routine manipulating this state, and may be instantiated several times. The extension of the existing function block by object-oriented features is a natural way of introducing object orientation to the IEC 61131. The features proposed in the current working draft are: methods, inheritance, interface abstraction [17]. Object-oriented programming (OOP) has demonstrated its capability for handling complex software development problems in an elegant way and for producing flexible, reusable software components. However, industrial control are not particular computer systems and then an object-oriented programming tool should satisfy the following requirements to be adopted in industrial automation [14]: Integration in a PLC development environment, e.g. I/O configuration, direct access to I/O signals; Multi-paradigm programming, i.e. object-programming should be optional to offer a stepwise and reversible transition to OOP; OOP by extension of the IEC, a small set of standard object-oriented features should be included in PLC programming, so that PLC programmers avoid a steep learning curve; Multi-lingual, OOP should be supported in all languages of the 61131-3 so that both textual and graphical languages, useful in programming sequences, can be used. At best of our knowledge, an object-oriented programming tool is available on the market in the form of the IEC-development environment CoDeSys V3n (see [1] for further details). Essentially, CoDeSys meets the above requirements and extends the FB construct to a class construct by the addition of methods, inheritance, and it introduces the INTERFACE-construct for the declaration of abstract FBs with polymorphic reference semantics.

2. Event based behavior. The International Electro technical Commission has developed a new standard called IEC 61499 [7]. It is a distributed control standard developed to provide an implementation independent standard based largely on methods already used in industry, that are part of the IEC 61131 standard. A good introduction to modeling controllers using IEC 61499 is provided
by [10], while [16] gives a practical introduction to using IEC 61499 controllers in industry settings. An application model in 61499 standards is a network of interconnected FBs. Thus, at the core of the standard there is the FB model that is here briefly recalled. FBs have clearly defined interfaces of event and data inputs and outputs. Event inputs are used to activate the block; event outputs are used to propagate events to other blocks.

2. 61131 Function Block revisited

2.1 Sequential Functional Charts

Sequential Functional Chart (SFC) is one of the most used IEC 61131-3 programming languages [8]. The basic components are: steps represented by a rectangular boxes, transitions represented by horizontal bars and oriented arcs represented by lines. The steps can be active or not; the state of a step is denoted by a Boolean variable which can be used to construct logical condition. Transitions have a firing condition associated. If all steps having output arcs to a transition are active and the condition associated to this transition is true, the transition fires: all the steps before it are deactivated and all steps after it are activated. The initial steps are represented by a double-boxed square. Actions that consist in simply setting the value of a Boolean variable or in executing a 61131 algorithm may be associated to each step: they represent the operations to perform in the system when the step is active. In fig. 1a) an example of SFC is given. START is the initial step. If REQ S1 is true, SERVICE 1 is activated and START is deactivated. When SERVICE 1 is active, the variable S1 is set to true.

![Figure 1. Execution Control SFC.](image)

2.2 A new FB model

The key concept used in this paper is a different use of FB model with respect to that of the 61131 standard. However, the FB model proposed in this paper is still fully 61131 compliant.

Three concepts are used to make 61131 FB programming near to event-based behavior:

- A different organization of input and output variables. As it is shown in fig. 2 input variables are assumed to be divided in event inputs, resource inputs, auxiliary data inputs, field data. The same occurs for output variables.

- The FB body can be programmed in any 61131 programming language, but it must always include a set of SFCs, called Execution Control SFCs (ECSFCs), which change their active steps according to the event inputs only. Such inputs are considered to be service requests. The allowed actions consist in setting the value of binary variables used to enable the execution of an algorithm. The end of the algorithm running produces an output event. Such outputs are considered to be complete-service events. The SFC in fig. 1a) activates the SFCs in fig. 1b) by setting to true the Boolean variable Si, which is associated to the condition of the unique output transition of the initial step of SFCi which represents the algorithm to execute to complete the service Si. Thus, the SFC in fig. 1a) acts as an ECSFC. The structure of ECS-
FCs can be generalized using hierarchical SFCs, i.e. when some SFCs may directly modify the marking of other SFCs. Hierarchy improves SFC modeling capabilities and solves some very important problems in the design of control algorithms [6]. A service can be implemented calling methods. Each method acts as event handler. Furthermore, instead of building variants of a function block by copying the function block and changing its implementation, FBs may be constructed by inheritance. A new FB inherits all variables and methods of the old function block but it may also define additional variables and/or methods.

- An algorithm is assumed to be associated to each service. A service can require a resource to be executed in addition to field and auxiliary data as usually. At this aim a set of input variables is reserved for resource availability. When a service execution terminates, the resources are released to be used for the activation of other service requests. A resource output variable represents something other FBs need to request a service to a certain FB. For example, the occupancy state of a conveyor is a resource output data of FB which controls a conveyor since other FBs, before requiring a moving service to it, must know if it is or not free. A resource input variable represents something a certain FB needs to know in order to request a service to another FB. For example, a FB which coordinates a set of conveyors, to transfer a pallet between two conveyors, must know if the destination conveyor is free before requesting the transfer service.

The advantages of this FB model are:

- A service, and so the associated algorithm, is executed only when it is explicitly invoked as in 61499. This allows overcoming different behavior of a FB due to the cyclically scanned program execution which is the core concept of 61131 architecture but, at same time, the code so designed is fully 61131 compliant. Indeed, also when suspension and forcing are used, the ECSFCs can be translated in any other 61331 programming languages [5].

- Different execution modes of a FB can be easily managed selecting them by input events.

### Figure 2. FB model

3. SoA implementation

In DES supervisory control theory it is assumed that a symbol is associated to each event and the behavior of a DES can be described in terms of generated sequences of events/symbols that form a language [12]. Consider for instance working cells composed by machines, conveyors and buffers: generated events can be the end of a working sequence, the arriving of a part at the end of a conveyor, the end of the loading of a part from a buffer; forced events can be the start of a working sequence, the loading of a part on a conveyor, the loading of a part from the buffer. Desired behaviors can be a correct machines and buffers loading sequence, avoiding buffer overflows, mutual exclusion when using shared resources. The FB model presented in Section 2.2 makes reasonable the abstraction of supervisory control theory that assumes events spontaneously generated by the physical process, useful for simplifying theory, is preserved and the implementation is not ambiguous (control and supervision concepts are not mixed).

Furthermore, thanks to the input and output organization of FB model, each device and so the whole physical process as controlled by FBs is seen as a service provider where the service started or service completed events are spontaneously generated. To design the supervisor all the results presented in the last years in the literature can be used and it is not the goal of this paper. For the sake of brevity,
supervisory control and Petri background are not recalled here. The reader can refer to [9] to find an introduction and many references on the topic. What must be here clarified is the matching between the controllable and uncontrollable events used in the supervisory control theory and the service requests and completion events used in SoAs.

Service requests are assumed to be controllable events, i.e. they are forced by the FBs but they are enabled by the supervisor. The ECSFCs under the initial state are ready to "force" the execution of all services but just the services enabled by the supervisor by means of service requests are performed. The service completion is an uncontrollable event since its occurrence cannot be disabled. This events are generated by FBs when the sequence responsible of a certain service has been completed.

In this paper it is proposed to use PNs as formal model but this is not mandatory. The empty boxes represent controllable events, while black bars are used to represent uncontrollable events. In fig. 3 an example of PN supervisor is presented: two sequences are specified; each sequence involves three services; two services (S1 and S5) require a common resource.

![Figure 3. An example of PN supervisor used to coordinate the service requests.](image)

The benefits of SoA in industrial automation context is twofold: if the body of a FB is modified but the set of service inputs and outputs remains the same, the supervisor does not change; if the specifications about desired sequences of services change, the FBs must not be modified.

4. Case Study

The effectiveness of the approach has been tested on a prototype of automated warehouse installed in our lab and shown fig. 4. The layout of the prototype is shown in fig. 5a). It consists of a set of bidirectional conveyors placed on four levels. The conveyors are used to move as well as to store pallets. Three lifts are used to move pallets from different levels.

![Figure 4. Prototype of automated warehouse](image)

![Figure 5. Automated warehouse prototype layout.](image)
The main goal of the control system is the transfer of a pallet between different locations, see for example 5b,c) where the paths of a pallet from location C1 at level 2 to C2 at level 3 or to C3 at level 1 are depicted. In general, to perform such transfers, a pallet must be moved along more devices according to a certain sequence. The term mission is used for such movement sequences. If a pallet must be moved from C1 to C3 or to C2, the use of the lift L2 is required. Each lift must be moved at level 0 after each movement.

It is assumed that an RFID reader is installed on the conveyor C1 and that each pallet is equipped with a RFID tag [13]. Depending on the value of the RFID tag a pallet on C1 must be moved to C2 or C3. A conveyor with a RFID reader becomes a smart device since it is able to provide movement but also information. SoA and OOP helps in programming the behavior of these devices, any service is seen as at same level and can be implemented properly calling more than one method as shown in the following.

The control of this system can be designed properly connecting FBs, which are requested to perform movement or information services. Then, to implement the service coordination a PN supervisor can be used since, as usual in practice, in this case the problem of supervising concurrent services or mutual exclusion in using sharing resources must be taken into account.

Figure 6. FB CONVEYOR

Figure 7. FB LIFT
A FB for each conveyor has been designed (see fig.6) which provides the following services:
LOAD FROM RIGHT (LR), load a pallet from right side;
LOAD FROM LEFT (LL), load a pallet from left side;
UNLOAD TO RIGHT (UR), unload a pallet to right side;
UNLOAD TO LEFT (UL), unload a pallet to left side;
STATE, detect the occupation of the conveyor.

The following input events are used:
REQ LR: request of service LR;
REQ LL: request of service LL;
REQ UR: request of service UR;
REQ UL: request of service UL;
REQ STATE: request of service STATE.

The following output events are used:
LROK: service LR completed;
LLOK: service LL completed;
UROK: service UR completed;
ULOK: service UL completed;
STATEOK: service STATE completed.

When a service is required to a device, it is important to verify that this can be done except when physical problem occurs. This is why, resource inputs and outputs have been introduced in our FB model. A unique resource data is needed for FB conveyor, the output resource data STATE (pallet on conveyor) to be sure that transfer of a pallet to (from) the conveyor are request when it is free (it has a pallet on board).

The auxiliary data output P DESTINATION is generated. It represents the destination of the pallet on the conveyor and it is an integer variable produced when the service REQ STATE is requested. The service is performed using the RFID reader. Notice that the use of this FB does not depend on the particular RFID reader used. If it is changed, the FB must be modified but the service call is always the same.

The following field data inputs are available:
PHOTOL, photocell at left end of conveyor;
PHOTOR, photocell at right end of conveyor;
RFID, content of the RFID label of the pallet on the conveyor.

The following field data outputs are generated:
GO LR, move conveyor from left to right;
GO RL, move conveyor from right to left;
READ, read the RFID label of the pallet on the conveyor.
A FB for each lift has been designed (see fig. 7). It is inherited from CONVEYOR and it provides the following additional services with respect to the FB CONVEYOR: MOVE i, move the lift at level i (i=1..4).

Hence, the following input and output events are used: REQ MOVEi: request of service MOVE i (i=1..4); MOVEiOK: service MOVE i completed (i=1..4).

While the following set of addition field input and output data events are available:
- S RAISE, lift is gone up by one level;
- S FALL, lift is going down by one level;
- LEVEL ZERO, lift at level 0;
- GO UP, move up the lift;
- GO DOWN, move down the lift;
- L CURRENT, lift current level.

In fig. 8 the program to implement the service LR is shown. This service requires the calling of two methods according to a sequence. Notice that the program is a SFC, but it is not a ECSFC.

For the sake of simplicity, assume that just two missions must be performed: transfer a pallet from C1 to C2 or from C1 to C3. In fig. 9 is shown the PN supervisor. Notice that to enable the supervisor a Boolean variable INPUT is supposed to be set to true by an higher level system to require to the supervisor the service of supervising the two missions. Notice that the supervisor enables the service requests so that a desired sequence is performed. The advantage to use a model is that of formally verifying a specification. In this example the PN supervisor is implemented as a FB, but this is not a rule. The supervisor can be implemented in any programming language and executed on a dedicated control device, not necessarily a PLC since it does not requires a direct interface with the physical process.
Finally, in fig. 10 it is shown the program in FBD obtained by properly connecting the FBs. For the sake of simplicity, input and output variables declaration has been omitted while FB instances declaration is shown to help the FBD comprehension. Notice that except for the FB implementing the supervisor, all the FBs are general and so reusable. The approach is absolutely modular.
5. Conclusions

Industrial automation is mainly based on PLC control system. PLC is now most programming language to IEC 61131 standard not ready to meet the new challenges of the widely distributed automation system. At present, the different solutions is using--from the industry have the ability and research: an expansion of the IEC 61131 including the object-oriented program design, using new standard IEC 61499 or implement monitoring. All of these solutions present different problems, this paper briefly recalls. To overcome these problems that some of the, a novel method based on monitoring and IEC 61131 could immediately put into use, to meet new requirements.

6. References


