A Programmable Logic Controller (PLC); Programming Language Structural Analysis

Yulia Kovalenko(✉)

National Aviation University, Kiev, Ukraine
yleejulee22@gmail.com

Abstract. Application of Programmable Logic Controller in the sphere of industrial automation substantially simplified technological processes control. New data exchange systems and new algorithms are being developed. It leads to enormous variety of comptrollers. We consider a programmable logic controller (PLC), a device that performs control of the physical processes of the algorithm written in it, oriented to work with devices developed through the input sensor signals and output signals to the actuators. PLCs are designed to work in real-time systems. Note that one of the advantages of the PLC system is modular. That is, the ability to combine and mix of types of input and output devices in a manner that best suits the application. Each of them differs by specific set of functions, unique construction and certain control language. In this article we describe classification of PLC, that can help to choose one, and also present the structural analysis of PLC programming languages.

Keywords: A programmable logic controller (PLC) · Programming languages · Real-time system · Diagram type language SFC

1 Introduction

Controllers may vary in several aspects, namely the manufacturing country, capacity and scope of use. The first factor lost its relevance some time ago, since the quality of controllers is increasing rapidly worldwide. In terms of capacity, PLCs are subdivided into nanocontrollers, small, medium, large and extra-large controllers. They differ by the number of I/O, network interfaces, memory, word length and main CPU speed. However, the most important factor to consider when selecting a programmable controller is, generally, the scope of its application:

- specialized controller with built-in functions. They feature a low capacity. These controllers already have an action program embedded, and you can only change its settings. This type of controllers is often able to implement various functions, and this determines a set of I/O modules. These PLC are commonly used in a small mechanical installation or a small gear;
- a controller to implement logical dependencies. This type of controllers is also called smart relay, and even the language of controlling such devices is similar to the relay circuits in many respects. It contains a large library of ready logic functions. It is also engaged in blocking of standard actuators. It is used mainly in the

R. Szewczyk and M. Kaliczyńska (eds.), *Recent Advances in Systems, Control and Information Technology*, Advances in Intelligent Systems and Computing 543, DOI 10.1007/978-3-319-48923-0_29

engineering and tank construction. A set of I/O modules is designed for digital channels;
- a controller for implementation of computing and logical functions. It is used almost everywhere thanks to a special generic nature. CPU can handle both logical and computational problems. Typically, such devices are not limited to a single control language, and another coprocessor is added for a better computational efficiency;
- crash protection controller. They are remarkable for fault tolerance, high availability and reliability, which is achieved by different methods of diagnostics and redundancy;
- telemechanical automation system controller. These controllers are generic too, but the scope of their application is quite narrow, namely supervisory systems of monitoring the objects distributed about the terrain. Here the most important thing is components for information data transmission at a distance via a wireless network.

Thus, we consider a programmable logic controller (PLC), a device controlling the physical processes according to an algorithm recorded therein, oriented to work with devices through a branched input of sensor signals and signal outputs to the actuators [1]. PLCs are designed to work in real-time systems. Let's note that one of the PLC advantages is modularity, i.e. an ability to combine the types of input and output devices in a way, which best suits the application [2, 3].

PLC structure can be subdivided into four parts, i.e. input-output modules, CPU, memory and terminal (Fig. 1).

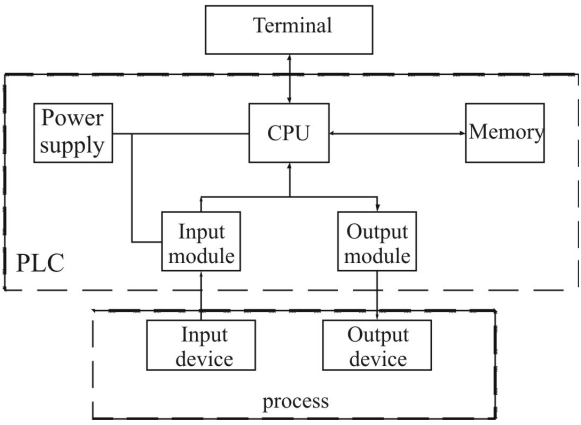

Fig. 1. Programmable logic controller (PLC) structure

2 Materials and Methods

Focusing on IEC 61131-3 [4–6], we determine the syntax and, for a smaller part, semantics of four programming languages for PLCs, as well as an aid for application structuring (SFC diagram-type language).

The first four languages include the LD ladder diagram language, FBD functional diagram language, ST textual high-level language and IL textual low-level language. LD, based on the ladder diagram, allows describing the logic functions. In FBD language, the functionality is presented as graphic blocks. ST textual language is close to the Pascal language, and operates the procedural, conditional and cycle operators. IL is a device-independent assembler-like language. While ST and IL are textual languages, and LD and FBD are graphic ones, SFC, in addition to its own syntax, can be used in conjunction with any textual or graphic language (Fig. 2).

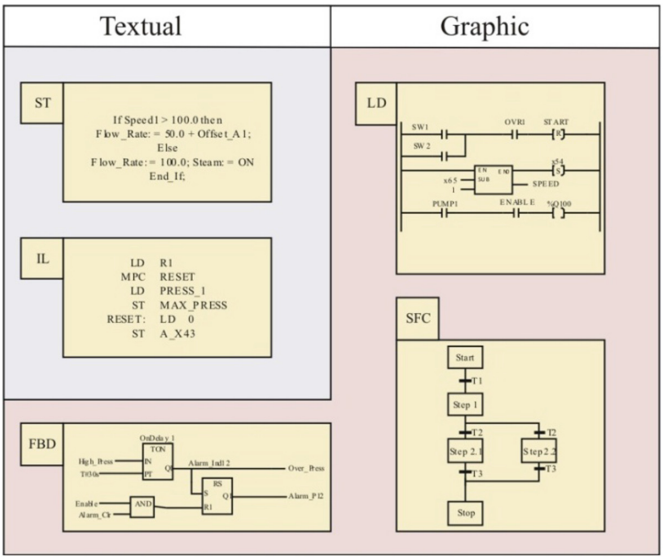

Fig. 2. Overview of IEC 61131-3 languages

SFC was created in order to structure the internal organization of PLC applications or function blocks. This graphical language called SFC (sequential function charts) is a generalized terminal machine containing the primitives describe serial, parallel and alternative lines of conduct. It allows dividing the PLC application (or function blocks) by the sets of interrelated steps and statuses. Each step corresponds to a set of actions, and each status is related to the status transfer condition.

Let's single out the principles of IEC 61131-3 languages:

– the entire application is subdivided into a set of functional elements, Program Organization Units (POU). These elements may be implemented using standard languages and consist of three categories [7]:
  - POU functions while running return only one value, defined by one of the standard types, function result and an arbitrary set of additional output data. These functional elements do not have a certain status (do not contain any information about the status), i.e. a function call with the same arguments always produces the same result;

  - POU functional units return one or more values as the result. The functional unit status is maintained run after run. It features a probabilistic aftereffect, so the call with the same arguments can return different results;
  - POU applications can be defined as a "logical connection of all elements and structures of the programming language required to process the selected signal to control the mechanism or process via a programmable controller" (IEC, 2003). Their definition and use is equivalent to functional units. They also can use the previous two POU categories as additional items.
- standard requires strict data typing. Specifying the data types can detect most errors in the application prior to its running;
- there are means to run various program fragments at different times, at different rates, and in parallel. For example, one program fragment can scan the terminal sensor 100 times per second, while the second fragment will scan the sensor with a rate of once every 10 s;
- to perform operations in a specific sequence triggered by time or events, a special Sequential Function Chart (SFC) language is used;
- the standard supports structures for description of heterogeneous data. For example, the pump bearing temperature, pressure and the "on-off" status can be described by a single "Pomp" structure, which can be transfer within the application as a single data element;
- standard provides for sharing of all five languages, so the most convenient language can be selected for each piece of the problem;
- the application written for a single controller can be transferred to any controller compliant with IEC 61131-3 [6].

So, let's analyze five languages of the IEC 61131-3 standard for PLCs. The use of each language in the PLC application involves two steps [8]:

- define the IEC 61131-3-based semantics;
- define the transition systems representing the language elements, and define the general rules of element layout in the application for ST and LD languages. This leads to definition of operational semantics.

The operational semantics consists of a transition system modeling the PLC running PLC cycle, plus one transition system for each language, representing the behavioral rules of the selected language, and several transition systems modeling the application elements. Modelling of behavioral rules depends on the chosen language.

SFC language structure is converted in the transition system calculating the application change algorithm. The approach of LD and ST languages is more successful when compiling the application in the transition system.

IL (Instruction List) is a textual low-level language. It is generic and often used as a common intermediate language into which other languages are translated. IL is a linearly-oriented language, its main advantage is the ease of study. IL can be programmed using any text editor. It is commonly used to solve small problems with a small number of branches and to write the most critical places in the application, as it allows creating the highly efficient and optimized functions.

At the moment, the third edition of IEC 61131-3 [9] declared the IL obsolete and undesirable for use.

The ST language uses a different approach. ST (Structured Text) is a high-level language, comprising a plurality of designs to assign the values to variables, call the functions and function blocks, write the expressions, conditional branches, choose the operators, and construct the iterative processes. This language is mainly intended to perform complex mathematical calculations, describe the complex functions, function blocks and applications. Its advantages over IL is a very concise formulation of programming tasks, clear program structure in the blocks of operators and major structures to control the running progress. ST algorithm is subdivided into several steps (operators) used to calculate and assign the values in order to control the run, call or exit from POU. In contrast to IL, ST can be defined by several lines or several operators, which may be written in one line.

LD (Ladder Diagram) language is based or the ladder flowcharts with horizontal links between the vertical power rails, executed sequentially. This programming language is developed mainly to process the Boolean signals (true/false). The buses link the LD network on the left and on the right. From the left bus, the current, controlled by "1" signal status, reaches all connected elements. Depending on their status, the elements either allow the current to proceed to the following elements, or interrupt the flow.

LD-network is described by the vertical and horizontal lines, as well as intersection points. The contact performs a logical operation based on the value from the incoming line and the required variable value. The logical operation type depends on the contact type. The value derived from the right connected line is the desired result.

However, it is hard to use the LD to implement complex algorithms, since it does not support routines, functions, encapsulation and other application structuring agents to improve the programming quality. These shortcomings make it difficult to reuse the software components, making the application long and complicated for maintenance. Another downside is that only a small part of the application fits the computer screen or operator panel during programming.

Despite these shortcomings, the LD language is one of the most common in the world [3], although it is used to program simple tasks only (Fig. 3).

FBD (Functional Block Diagram) is a graphical programming language using the functional block diagrams to submit an application to the PLC. It is based on the basic

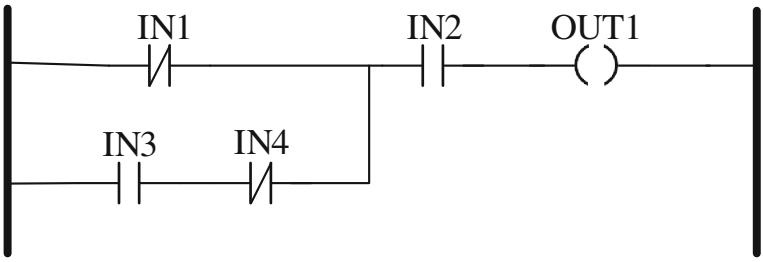

Fig. 3. Example of application in the Ladder Diagram language

block diagrams caused by a graphical representation of electrical circuits and basic language elements, i.e. blocks. Block is a subroutine, a function, or a function block. The functional blocks encapsulate the data and methods, like the object-oriented programming languages, but do not support inheritance and polymorphism. Another block feature is parameterization of inputs and outputs, which can be represented by various types. For example, the ADD block function may have from two to any admissible number of inputs, and a summation of all inputs will be fed to the output.

FBD borrows the symbols of Boolean algebra and, as Boolean symbols have inputs and outputs, which can be connected to each other, FBD is more efficient to represent the structural information than the ladder diagram language. Also, in addition to the basic elements of graphic languages, FBD has some unique elements (Fig. 4).

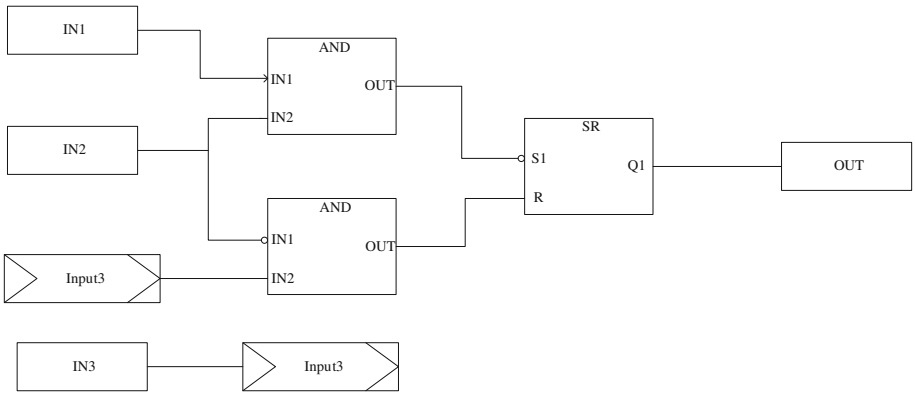

Fig. 4. Example of an application in the Functional Block Diagram language

The diagram-type SFC (Sequential Function Chart) language allows representing a POU application or a PLC functional block using the graphical and textual notation system. SFC is essentially an aid to structure the applications by breaking the main control application branch into smaller components and monitoring of their implementation. Its basis is the mathematical formalism of Petri nets (PNs), which allows describing the processes in the form of bipartite directed graphs [10] (Fig. 5). The graphical representation allows clearly defining the application running progress, and allows the design of serial and parallel application processes.

The operation of any T_j transition within the marked network leads to a change in the marking. Running of the smaller program components (e.g., processes or branches) depends both on conditions determined by the application, and on behavior of the input and output data. The components are programmed directly in one of the other IEC 61131-3 languages. The processes with a stepper behavior are particularly suitable for programming in SFC.

The first level of structuring in SFC is a network made up of elements called steps and statuses. Step can be active or inactive. When it is active, the required commands are run until the step becomes inactive. The step status substitution is determined by the

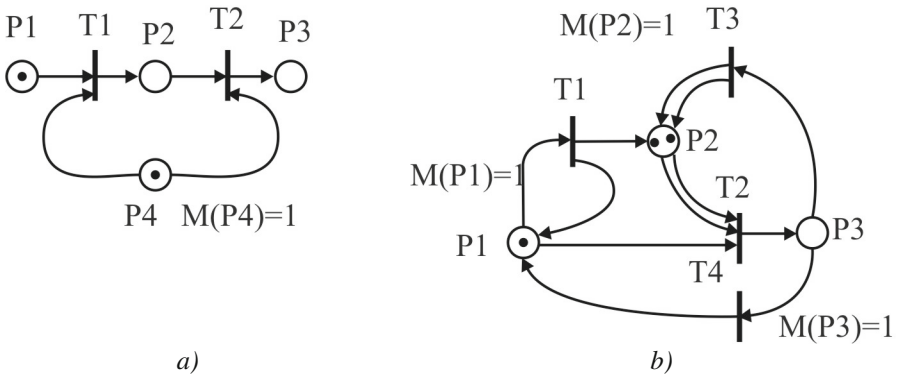

Fig. 5. Process presentation in the form of bipartite graphs. Where: P_i is positions, T_j is transitions, $M(P_i)$ - markings

transition status, which is a logical expression. If the transition condition becomes true, the next step becomes active and the previous step is deactivated. With the transition change, the "active" property moves from an active step to its successor or successors; such movement forms a network. This property may be divided when the parallel branches are run, and then restored after completion of the branch run.

IEC-actions in steps have special classifiers determining how they are run within the step: cyclic running (N), a one-time running (P), etc. There is a total of nine qualifiers, including the classifiers with saving (S), delayed (D) and time-limited (L) actions (Fig. 6).

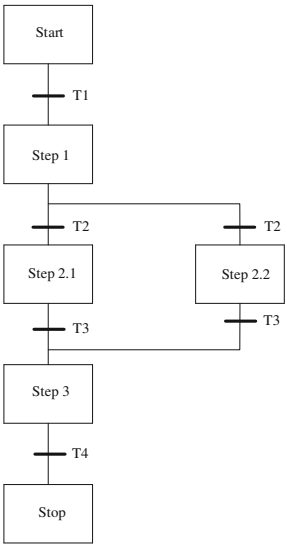

Fig. 6. A set of Sequential Functional Chart network steps combined by transitions

3 Results and Discussion

After analysis of the programming languages, we conclude that the controlling systems operating in real-time logic processes can be programmed on SFC. LD, FBD and IL are the programming languages suitable for formulation of the main action and for operating systems, which can be described by simple logical operations or logic signals. ST language can be used mainly to create the software modules with a mathematical context, for example, to describe the control algorithms. Perhaps, the most powerful and versatile tool from the IEC 61131-3 languages is a SFC + ST combination. SFC language graphics facilitates the language learning. The presence of common roots with Petri nets partially eliminated the synchronization and parallelism issues. Programming of operations with analog and logic variables is comfortable enough due to the use of the textual Pascal-like ST language. The command flow control does not cause problems. A significant advantage of the approach is event-based nature, naturally supported through a "step-transition" mechanism. The application debugging can be facilitated by visual tracing of the control flow. The weaknesses of SFC language (like Petri nets) is an abstraction and structuring [11], which, as in previous cases, adversely affects the difficulties of programmable algorithms and their quality (reliability and maintainability). SFC + ST approach is inferior to FBD and LD in terms of convenient programming of the algorithm parallelism and thus is more suitable to program the linear algorithmic sequences.

4 Conclusions

A typical example of use is combined algorithms of the logic and analog control. The programming experts combining the knowledge of programming languages and algorithmic peculiarities of the automated process are proposed to be the users [9].

Where the PLC or PLC software permits, all created applications or application parts have to be simulated before launch. This allows detecting and eliminating the errors at an early stage, which would then reduce the complexity and cost of such applications.

References

1. Korobiichuk, I.: Mathematical model of precision sensor for an automatic weapons stabilizer system. Measurement **89**, 151–158 (2016)
2. Korobiichuk, I., Podchashinskiy, Y., Shapovalova, O., Shadura, V., Nowicki, M., Szewczyk, R.: Precision increase in automated digital image measurement systems of geometric values. In: Jabłoński, R., Brezina, T. (eds.) Advanced Mechatronics Solutions. AISC, vol. 393, pp. 335–340. Springer, Heidelberg (2016). doi:10.1007/978-3-319-23923-1_51
3. Korobiichuk, I., Shostachuk, A., Shostachuk, D., Shadura, V., Nowicki, M., Szewczyk, R.: Development of the operation algorithm for a automated system assessing the high-rise building. Solid State Phenomena **251**, 230–236

4. Bonfatti, F., Monari, P.D., Sampieri, U.: IEC 1131-3 programming methodology. Software engineering methods for industrial automated systems. CJ International Editions (1997). ISBN 2-9511585-0-5
5. Ohman, M., Johansson, S., Arzén, K.E.: Implementation aspects of the PLC standard IEC 1131-3. IFAC Control Engineering Practice 123 **6**(4), 547–555 (1998)
6. Lewis, R.W.: Programming industrial control systems using IEC 113-3 Revised edition, 329 p. The Institution of Electrical Engineers, London, UK (1998)
7. Barbosa, H., Déharbe, D.: Formal verification of PLC programs using the B method. In: Proceedings of the Third International Conference on Abstract State Machines, Alloy, B, VDM, and Z, pp. 353–356 (2012)
8. De Smet, O., Couffin, S., Rossi, O., Canet, G., Lesage, J.-J., Schnoebelen, Ph., Papini, H.: Safe Programming of PLC Using Formal Verification Methods. Ecole Normale Suprieure, Chaire De Fabrications, France (2000)
9. IEC 61131-3:2013 Programmable controllers - Part 3: Programming languages
10. Anisimov, N.A., Golenkov, E.A., Kharitonov, D.I.: Composition approach to development of parallel and distributed systems based on Petri nets. Programming (6) (2001)
11. Ziubin, V.E.: PLC programming: IEC 61131-3 languages and possible alternatives. Ind. ACSs Control. (11), 31–35 (2005)