25

Object-Oriented Development Methodology for PLC software

O. Durán. and A. Batocchio
orlando@fem.unicamp.br batocchi@fem.unicamp.br
Depto. Eng. de Fabricação, FEM UNICAMP FAX (0192) 393722
CP 6122 CEP 13083-970 Campinas (SP) Brazil

Abstract

This paper reports the application of an Object-Oriented Development Methodology for specifying Programmable Controllers Software. This methodology, called OOST, allows the user to specify the control logic in a natural manner, using a collection of objects that represent devices and other machines in an actual manufacturing system. The object-oriented specification technique may be used in the whole control software life cycle. It could be considered as a requirement definition language or an implementation one, since the OOST may be used as an input to an automatic PLC program generation system. A brief literature review is presented and the implementation details are discussed. In the final part of this paper some future improvements and remarks are given.

Keywords

PLCs software, Object-Oriented Development, Automatic Program Generation

1 INTRODUCTION

Programmable Logic Controllers are widely used for sequential operation control, specifically in the discrete manufacturing industry. This fact is due to the advantages and improvements obtained when compared with fixed logic systems and relays-based systems. The improvements are refereed to flexibility, safes and low maintenance costs, and the reduction in start up and operation times (Cox, 1986).

The growing up success of this equipment brought a high diversity of PLCs into the shop floor. Each day more and more manufacturers enter to the market, offering new solutions, sweeping a wide range of possibilities and features. The principal features that the manufacturers outline for winning the high competitions' levels are: throughput, Input/Outputs points, programming facilities and languages, etc.

However, there is a trouble that begins to manifest when a PLC user purchases different manufacturers' equipment. Each one of these manufacturers incorporates a proprietary programming language to their products. This fact leads to serious difficulties to the user, such as: no chance for reusability, no program sharing among solutions and so

L. Camarinha-Matos et al. (eds.), *Balanced Automation Systems*

on. A solution for this problem is to have a programming specialist for each one of the programming languages used into the shop floor (Halang, 1992)

Another trouble related to PLCs software development is the extended life cycle. It is very difficult pass from one phase to another in a smoothly and natural manner, mainly because of the traditional approaches do not have been realized as a manufacturing software, but as data intense applications development tools.

Manufacturing Engineers realize the manufacturing domain as composed by different entities, such as machine tools, fixtures, pieces, material handling devices, sensors, actuators, etc. Each one of these entities has associated set of attributes and capabilities constituting abstract constructions called objects. Through this abstraction process a system definition task is simplified and made in very natural manner.

This paper reports an approach to developing PLCs software using the Object Oriented Paradigm. This approach, that is being widely used in Software Engineering, is used here for automated program generation for a PLC. The automatic generation process is performed from a textual and semi-structured description of the control logic for an automated manufacturing cell (Durán, 1993).

2 BACKGROUND

There are some researches in the literature aiming at the development of manufacturing systems modeling and control software. Menga and Morisio (1989) defined a specification and prototyping language for manufacturing systems. The language is based on high-level Petri-nets and is object-oriented. The approach uses graphical and textual tools for defining the components of the system and their behaviors. The language integrates two formalism with the object-oriented development paradigm. The first formalism is the hierarchical box and arrow graphical formalism. The second one is the Petri-net graphical formalism for the detailed specification of the control flow in objects. According the authors the use of this language allows the simulation of manufacturing systems, hierarquically structuring them and enriching the library with the new objects.

Another initiative (Boucher, 1992) addresses the definition of an interface between manufacturing system design and controller design. The high-level design methodology enhances communications between manufacturing system designers and controller designers. It also allows the automatic control code generation from that design. The high-level design methodology is based in the IDEF0 methodology, created under the development of USAF, and generates a Petri Net representation of the control logic through the use of a rule-based interpreter.

A similar approach is given by Roberts and Beaumariage in (1993) that present a methodology to design and validation of supervisory control software specification. This methodology is based on a network representation schema. The networks are composed by nodes and arcs. Messages are passed between the nodes, resembling the object-message paradigm. A version of this control specification system have been coded on a Texas Instruments Explorer and other platforms.

A textual specification technique for PLC software generation was reported by Bhatnagar and Linn (1990). The generator uses a user-defined control process specification and initially converts it into a standard control logic specification and finally

generates an executable task code program for the PLC specified. The bidireccionality of the code is the main virtue of this system, providing total transportability of programs from a PLC to another.

Halang and Krämer (1992) presented an interactive system with a graphical interface for constructing and validating PLC software. It combines the Function Block Diagram with a graphical language. The approach emphasizes the description of composite PLC software from a library of reusable components and may be considered as an object-oriented approach.

Finally, Joshi, Mettala and Wysk (1992) presented a paper where a systematic approach to automate the development of control software for flexible manufacturing cells called CIMGEN. The specification is based on Context Free Grammars (CFGs) providing a formal basis for control strategies descriptions. CIMGEN generates automatically control software for workstation and cell control levels. This automated code generation is not totally satisfactory, since the generated code requires hand manipulation for completion.

There are two other papers that report Object-Oriented approaches for modeling manufacturing systems. Mize, Bhuskute, Pratt and Kamath (1992) relates the results obtained in exploring alternative approaches to the modeling and simulation of complex manufacturing systems. These results argue that is necessary a paradigm shift in developing models for manufacturing systems. Through this paradigm shift the system planner can now define models through the use of building blocks, called objects. The authors assert that this approach is an strategic opportunity for the fields of industrial engineering, operations research/management science and manufacturing systems engineering. Joannis and Krieger (1992) reported an Object-Oriented methodology for specifying manufacturing systems. The specifications are made by building successive models, each containing more details than the previous one. The desired behavior of the system is described using a set of concurrent cooperating objects and the behavior of each object is defined through the use of Communicating Finite State Machines (CFSM). According to the authors these CFSM allow the execution of the specifications. The technique has a text format.

3. MANUFACTURING CELLS CONTROL

Manufacturing systems are sets of different subsystems all working together and coordinately to get the expected results. The design of these manufacturing systems and the development of manufacturing systems controllers has become more closely linked as the manufacturing environment has become more automated (Boucher, 1992). Controller design addresses issues of communication, controller logic, sequencing, error handling and programming of programmable devices.

Usually the control of a manufacturing systems is made up in different levels of abstractions. (Menga, 1989) identified four levels to perform the manufacturing control, these levels are: plant leveel, shop level, cell level and macjine level. These levels of abstractions have different time scales, event types and decision kinds. The scope of this paper is the Manufacturing Cell control and the definition of the cell controller. The cell controller is responsible for the sequencing of activities within a cell. It is necessary to

define the functions related to the major operations performed for each one of the elements that make up the cell. The control at cell level is normally implemented using PLC or combinations of PLC and a cell host computer or a factory floor computer. Programming a PLC is an error prone task, where the developed programs are very difficult to read, understand and maintain. Hence there are many efforts to define a high level programming standard to simplify the program definition and the total PLC software life cycle. There are different approaches for programming a PLC. Each one of the manufacturers incorporate one or more types of these approaches to their equipment. The most common methods for programming a PLC are:

- Ladder Diagrams
- Instructions Lists
- Structured Texts
- Sequential Function Charts
- Function Block Diagrams

4 THE APPROACH

The system that is being developed allows the programmer specify a new application control program through the use of a object oriented specification technique (OOST). This specification technique may be used from the requirement description phase to the implementation phase (Booch, 1986), since it may be used as an specification input language for an automatic PLC software generator. The development of PLC software using the object oriented specification technique can shorten the life cycle phases, even some phases are no more needed, because they are contained within the previous phases (Hodge, 1992).

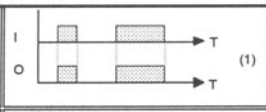
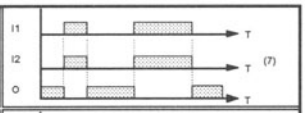
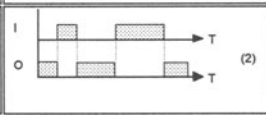
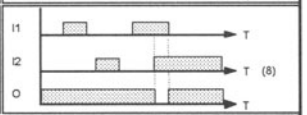
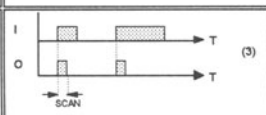
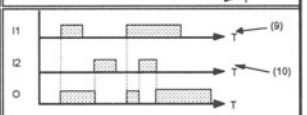
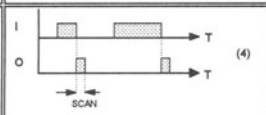
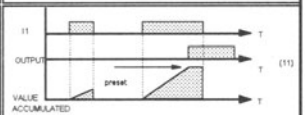
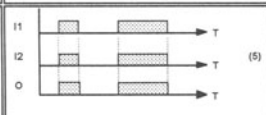
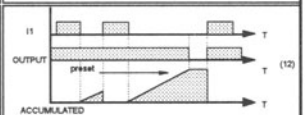
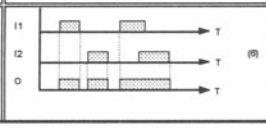
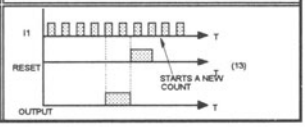
The methodology when applied in the requirement analysis phase allows the natural contact between the programmer/analyst and the final user of the system. Hence, the manufacturing engineer that is in charge to project the automated system can communicate his needs and requirements with the programmer/analyst using a common language, based on abstract constructions (objects), avoiding misunderstandings and excessive documentation.

To make up the specification technique, an exploratory survey among various PLC programming languages was made (Allen Bradley, 1993; Weg, 1991; Hitachi, 1994; Modicon, 1991). This survey aimed at defining a sufficient collection of basic operations supported by any PLC performing sequence control tasks, table 1 shows the set of basic operations considered to define the OOST.

Besides the instructions showed by table 1, there is a set of instructions for data manipulation and arithmetic operations. These instructions were also considered by our research and OOST supports them. Unlike of these operations OOST does not support complex instructions, such as PID functions and Fuzzy Logic-based operations, because they are out of the scope of our research. The basic operations were grouped into five categories, considering types of devices state changes. These five categories are:

- Momentarely state changes caused by the beginning or ending of an event.
- Permanent state changes caused by the beginning or ending of an event.
- State changes caused by the existence of an event during a given time.
- State changes caused by the existence of an event a given number of times.
- State changes caused and mantained during the existence of an event.

Table 1 Basic PLCs' operations that make up the OOST

Normally Open		Serial connection of normally closed contacts	
Normally Closed		Parallel connection of normally closed contacts	
Positive Transition		Latch and Unlatch coils	
Negative Transition		On delay timer	
Serial connection of normally open contacts		Off delay Timer	
Parallel connection of normally open contacts		Counter	

Next, the structures needed for describing each one of these categories was analyzed. Thus a set of textual sentences making up a description language was defined. This set of sentences was translated into a Definite Clause Grammar. Next this grammar was written using Prolog code. The following shows part of this grammar:

```
sentenças    ::= sentença, [';'], sentenças.
sentenças    ::= [].
sentença     ::= conjuntoA1, [','], oração_subor.
conjuntoA1   ::= conjuntoA1i.
             ::= conjuntoA1ii.
             ::= conjuntoA1iii.
conjuntoA1i  ::= conjunçãoA1i, agente, verboA1i.
conjuntoA1ii ::= conjunçãoA1ii, agente, verboA1ii.
conjuntoA1iii::= conjunçãoA1iii, agente, verboA1iii.
verboA1i     ::= verbo_aux_A1ia, verbo_inf.
             ::= verbo_aux_A1ib, artigo, verbo_sustantivado.
             ::= verbo_aux_A1ic, prepos_art, verbo_sustantivado.
verboA1ii    ::= verbo_aux_A1iia,
verbo_inf.   ::= verbo_aux_A1iib, artigo, verbo_sustantivado.
```

```
            ::= verbo_aux_A1iic, prepos_art, verbo_sustantivado.
verboA1iii  ::= verbo_aux_A1iiia, verbo_inf.
            ::= verbo_aux_A1iiib, artigo, verbo_sustantivado.
            ::= verbo_aux_A1iiic, prepos_art, verbo_sustantivado.
conjunçãoA1i::= ['assim que'].
            ::= ['quando'].
            ::= ['se'].
            ::= ['sempre que'].
            ::= ['depois que'].
            ::= ['logo que'].
            ::= ['uma vez que'].
conjunçãoA1ii::= ['caso'].
            ::= ['desde que'].
            ::= ['contanto que'].
conjunçãoA1iii::= ['dado que'].
            ::= ['visto que'].
            ::= ['já que'].
            ::= ['uma vez que'].
sentença    ::= oração_subor, conjuntoB1.
conjuntoB1  ::= conjuntoB1i.
            ::= conjuntoB1ii.
conjuntoB1i ::= conjunçãoB1i, agente, verboB1i.
conjuntoB1ii::= conjunçãoB1ii, verboB1ii.
verboB1i    ::= verbo_aux_B1ia, verbo_inf.
            ::= verbo_aux_B1ib, artigo, verbo_sustantivado.
            ::= verbo_aux_B1ic, prepos_art, verbo_sustantivado.
verboB1ii   ::= verbo_aux_B1iia, verbo_inf.
            ::= verbo_aux_B1iib, artigo, verbo_sustantivado.
            ::= verbo_aux_B1iic, prepos_art, verbo_sustantivado.
conjunçãoB1i::= ['a não ser que'].
            ::= ['a menos que].
            ::= ['até que'].
conjunçãoB1ii::= ['salvo se'].
sentença    ::= conjuntoC1, [','], oração_subor.
conjuntoC1  ::= conjuntoC1i.
            ::= conjuntoC1ii.
conjuntoC1i ::= conjunçãoC1i,agente, negação verboC1i.
conjuntoC1ii::= conjunçãoC1ii, agente, negação, verboC1ii.
verboC1i    ::= verbo_aux_C1ia, verbo_inf.
            ::= verbo_aux_C1ib, artigo, verbo_sustantivado.
            ::= verbo_aux_C1ic, prepos_art, verbo_sustantivado.
verboC1ii   ::= verbo_aux_C1iia, verbo_inf.
            ::= verbo_aux_C1iib, artigo, verbo_sustantivado.
            ::= verbo_aux_C1iic, prepos_art, verbo_sustantivado.
conjunçãoC1i::= ['contanto que'].
            ::= ['caso'].
            ::= ['desde que'].
conjunçãoC1ii::= ['dado que'].
            ::= ['visto que'].
            ::= ['já que'].
            ::= ['uma vez que'].
```

Finally, a rule-based interpreter was written. This task was made just translating the grammar into Prolog clauses.

5 USING THE SYSTEM

The user/programmer describes any manufacturing situation using OOST and a hierachy of objects that represent a specific manufacturing domain. The manufacturing system behaviour is described through a set of textual sentences ruled by the OOST grammar. Finally the description of the system is analysed by the second module of the system called Interpreter. The interpreter performs lexical and syntactical analysis of the specifications made by the user and to provide as an output an intermediate code of the control logic. This rule-based interpreter supports the object-oriented paradigm. That is, objects representing actual manufacturing devices may be defined, and these objects may be structured in a hierarchy using the concept of superclasses and classes. Through that, a new object class or object instance can inherit attributes and capabilities of its object superclass. It is the main element of the paradigm, that allows models reusability. The intermediate code, generated by the interpreter, is used as an input for an automatic PLC program generation module. The general structure of this system is shown in figure 1.

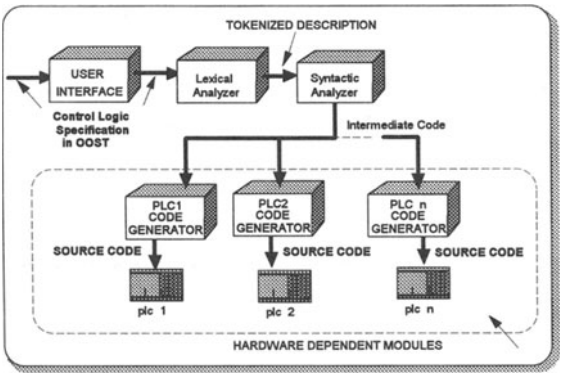


Figure 1 System Structure

The windows-based user-interface was written in Visual Basic. The other modules was written in Arity Prolog, using the concept of DCG.

6 CONCLUSIONS

This specification technique may be considered as an efficient means for performing an informal requirement description, and from it, to obtain in an automated manner the source code for an specific PLC, ready for downloading within PLC memory.

The approach allows to create object library for accelerating new applications developments and fomenting software reutilization (Shaw, 1984). This fact leads to important improvements in productivity and reliability of PLC programming tasks.

7 REFERENCES

Allen Bradley (1993) PLC-5 Programming and Operation Guide.

Bhatnagar, S. and R.L.Linn (1990) Automatic Programmable Logic Controller Program Generator with Standard Interface. *Manufacturing Review*, Vol. 3 No. 2,98-105.

Booch, G. (1986) Object Oriented Development. *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 2, February, 211-221.

Boucher T.O. M.A.Jarafi (1992) Design of a Factory Floor Sequence Controller from a High-Level System Specification. *Journal of Manufacturing Systems*, Vol.11, No. 6, 401-417.

Durán, O. and A. Batocchio (1993) A high-level object-oriented programmable controller program interface. *ISIE'94, Proceedings of the IEEE International Symposium on Industrial Electronics, Santiago, Chile, May, 1993.*

Cox, B,J. (1986) *Object Oriented programming: an evolutionary approach.* Addison-Wesley Reading, Massachusetts.

Halang, W.A. and B.Krämer (1992) Achieving high integrity of process control software by graphical design and formal verification. *Software Engineering Journal*, January, 53-64.

Hitachi (1993), Programmable Controller H-200, Operation Manual.

Hodge, L.R. and M.T.Mock (1992) A proposed object-oriented development methodology. *Software Engineering Journal*, March, 119-129.

Joannis, R. and M. Krieger (1992) Object-oriented approach to the specification of manufacturing Systems. *Computers Integrated Manufacturing*, Vol.5 No. 2, 133-145.

Joshi, S.B., E.G.Mettala and R.A.Wysk (1984). CIMGEN- A Computer Aided Software Engineering Tool for Development of FMS Control Software. *IIE Transactions*. Vol.24, No.3, July, 84-97.

Menga, g. and M.Morisio (1989) Prototyping Discrete Part Manufacturing Systems. *Information and Software Technology*. Vol. 31, No.8, 429-437.

Mize, J.H., H.C.Bhuskute, D.B. Pratt and M.Kamath (1992). Modeling of Integrated Manufacturing Systems using an Object-Oriented Approach. *IIE Transactions*, Vol. 24, No. 3, 14-26.

Modicon 984 (1991) Programmable Controller Systems Manual.

Shaw, M. (1984) Abstraction Techniques in Modern Programming Languages. *IEEE Software* October, 10-26.

WEG (1991) Programmable Controller A080, Programming Manual.

Acknowledgments: This project is being sponsored by the Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) e pelo Fundo de Apóio ao Ensino e Pesquisa (FAEP) of UNICAMP.

Orlando Durán received his B.S. degree in Industrial Engineering from Universidad de Santiago (Chile), and his M. Sc. degree in Mechanical Engineering from Universidade Estadual of Campinas (Brazil). He is currently performing his Doctoral studies. Research areas of interests are Automation, Object Oriented Modelling and Artificial Intelligence.

Antonio Batocchio received his B.S. and M.Sc. in Mechanical Engineering from Universidade Estadual de São Paulo (São Carlos, Brazil), and his Doctor degree from Universidade Estadual de Campinas (Brazil) in 1991. Currently he is a Visitant professor at the Mechanical Engineering Department of the Minnesota State University (USA). Research areas are interests of Automation, Manufacturing Cell Technology, Simulation and Costs.