1

Object-Oriented Modeling, Simulation and Automatic Generation of PLC Ladder Logic

Kwan Hee Han
Gyeongsang National University
Republic of Korea

1. Introduction

Most enterprises are struggling to change their existing business processes into agile, product- and customer-oriented structures to survive in the competitive and global business environment. Among their endeavor to overcome the obstacles, one of the frequently prescribed remedies for the problem of decreased productivity and declining quality is the automation of factories (Zhou & Venkatesh, 1999).

As the level of automation increases, material flows and process control methods of the shop floor become more complicated. Currently, programmable logic controllers (PLC) are mostly adopted as controllers of automated manufacturing systems (AMSs), and the control logic of PLC is usually programmed using a ladder diagram. More recently, manufacturing trends such as flexible manufacturing facilities and shorter product life cycles have led to a heightened demand for reconfigurable control systems. To cope with these challenges, a new effective and intuitive method for logic code design and generation is needed.

However, currently there are no widely adopted systematic logic code development methodologies to deal with PLC based control systems in the shop floor. So, the control logic design phase is usually omitted in current PLC programming development life cycle though it is essential to reduce logic errors in an earlier stage of automation projects before the implementation of control logic. Moreover, fast customer requirement changes requires flexibility of manufacturing system. To deal with these frequent configuration changes of modern manufacturing systems, it is required that logic code can be generated automatically from the design results without considering complicated control behavior.

To generate error-free ladder code, it is also essential to validate the designed control logic of an AMS in an effective way. Among many validation methods, computer simulation methods are widely used because mathematical formalisms have a problem of solution space explosion as the size of system increases. However, since current simulation methods have mainly focused on the overall performance evaluation of manufacturing systems such as factory layouts, resource utilization, and throughput time, they have limitations with regard to the modeling capabilities of detail logic for the input/output signal-level control of AMS. Therefore, current PLC ladder programming practices require a more integrated way to design, simulate, and generate the ladder control logic.

The main objective of this chapter is to propose an object-oriented (O-O) ladder logic development framework integrating design, validation and automatic generation of ladder

Source: Programmable Logic Controller, Book edited by: Luiz Affonso Guedes,
ISBN 978-953-7619-63-3, pp. 170, January 2010, INTECH, Croatia, downloaded from SCIYO.COM

logic using extended UML (Unified Modeling Language). Proposed framework, as depicted in Figure 1, consists of three parts: first part deals with UML design of PLC-based control system. O-O design model consists of three models: functional model, structure model and interaction model. Second part is concerned with O-O simulation method for validating designed ladder control logic. By using the results of O-O design model, an O-O simulation model is constructed and is executed. During the execution of simulation model, factory automation (FA) engineers can evaluate the system performance and validate the PLC control logic simultaneously. Last part deals with automatic generation of ladder code from the validated design result. In order to show the applicability of proposed method, an UML-based tool for the design and generation of ladder code is also developed. Proposed framework facilitates the generation and modification of ladder code easily within a short time without considering complicated control behavior to deal with current trend of reconfigurable manufacturing systems.

The rest of the chapter is organized as follows. Section 2 reviews related works. Section 3 describes UML design of control logic. Section 4 deals with O-O simulation of designed PLC-based control system for validating the correctness of control logic. Section5 describes the automatic generation and verification method of ladder logic. Finally, the last section summarizes results and suggests directions for future research.

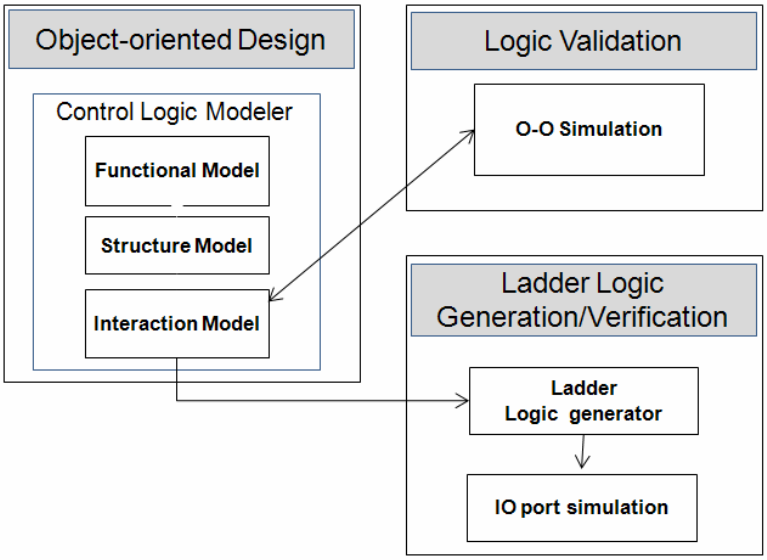

Fig. 1. Proposed object-oriented ladder logic development framework

2. Related works

In order to improve current PLC programming practices, significant efforts have been made in researches on object-oriented technologies in manufacturing systems. O-O modeling has been mainly used as a method for the analysis and design of software system. Recently, it is presented that O-O modeling is also appropriate for the real-time system design like an AMS as well as the business process modeling.

Several researches were made regarding the O-O modeling methods for the manufacturing system: Author of this chapter proposed AMS modeling framework called JR-Net (Job-Resource relation Net), which consists of a layout model, a functional model, and a control model for the O-O simulation of AMS (Choi *et al*. (1996), Park *et al.* (1997)). But, since this work placed emphasis on the supervisory control level rather than the device control level in the control model, it did not presented the modeling results of device level control. An O-O method for the design of automation system was proposed (Calvo *et al.,* 2002), but it only showed the static structure comprised of a class diagram and a use case diagram. An UML modeling of AMS and its transformation into PLC code was proposed (Young *et al.,* 2001), but it did not presented the method of PLC code generation. An UML modeling of flexible manufacturing system and its simulation implementation was proposed (Bruccleri & Diega, 2003), but it restricted the control level to the supervisory control level.

Among researches about design and validation tools for the PLC control logic, a simulation method integrating plant layout sub-model and control sub-model, and also a PLC code generation from simulation result was proposed (Spath & Osmers, 1996), but it omitted details of generation procedure. A procedure of control logic design was proposed by using IEC function block diagram (FBD) model, its transformation into Petri net, the validation of control logic using SIMULINK simulation system, and C code generation (Baresi *et al.,* 2000). But, it confined their modeling scope to simple control logic which can be represented by FBD. Author of this chapter developed O-O design tool based on the extension of UML and showed usefulness of O-O design and simulation approach to ladder logic development (Han & Park (2007a), Han *et al.* (2007b)).

In the area of automatic ladder logic generation method, there exist mainly three approaches as follows: First approach is Petri net-based (Peng & Zhou (2004), Lee *et al.* (2004), Frey & Minas (2001), Taholakian & Hales (1997)). Second approach is finite state machine-based (Jack (2007), Manesis & Akantziotis (2005), Sacha (2005), Liu & Darabi, (2002)). Last approach is flow chart-like-based (Jack (2007), Hajarnavis & Young (2005)). Among three approaches, first and second approaches have a state explosion problem when complexity of control logic increases.

The third approach is relatively easy to use by its sequential and intuitive nature to control logic programmers. However, the result of ladder code generated by the third approach proposed by Jack (2007) is different from the code directly written by FA engineers due to its automatic generation features. Therefore, it is not natural to FA engineers and revealed difficulties to understand the generated ladder code. Research about functionalities of Enterprise Controls commercial package of Rockwell Automation was presented, in which FA engineers design the ladder logic in the form of flow chart within Enterprise Controls, and ladder code is generated automatically (Hajarnavis & Young, 2005). However, it did not show how the ladder code is generated. Proposed generation method in this chapter belongs to the third category, in which ladder logic code is generated from the extended UML activity diagram which is a kind of flow chart.

3. Object-oriented design of control logic

The most typical features of O-O modeling techniques include the interaction of objects, hierarchical composition of objects, and the reuse of objects (Maffezzoni *et al.,* 1999). O-O design for ladder control logic is conducted based on system specifications such as drawings and problem descriptions. During the design phase, FA engineers develop three models for

describing the various perspectives of manufacturing systems: 1) a functional model for representing functional system requirements of AMS, 2) a structure model for representing the static structure and relationships of system components, and 3) an interaction model for representing the dynamic behavior of system components.
A functional model is constructed using an UML use case diagram in which each functional requirement is described as a use case. A use case diagram describes a top-level system view. A PLC as a plant controller is represented by a 'system' element, and input or output part of PLC such as a sensor, actuator, and operator is represented by an 'actor' element of a use case diagram. Since the UML stick man icon of 'actor' is not appropriate for representing the resource of AMS, new icons are introduced in a functional model using UML stereotype property. Therefore, in the extended UML use case diagram, as depicted in Figure 2, four types of actor (i.e., operator, actuator, sensor and MMI) are newly used instead of standard stickman symbol. PLC input parts such as sensor and operator are located at the left side of 'system' symbol, and PLC output parts such as actuator and MMI (Man Machine Interface) are located at the right side of 'system' symbol.

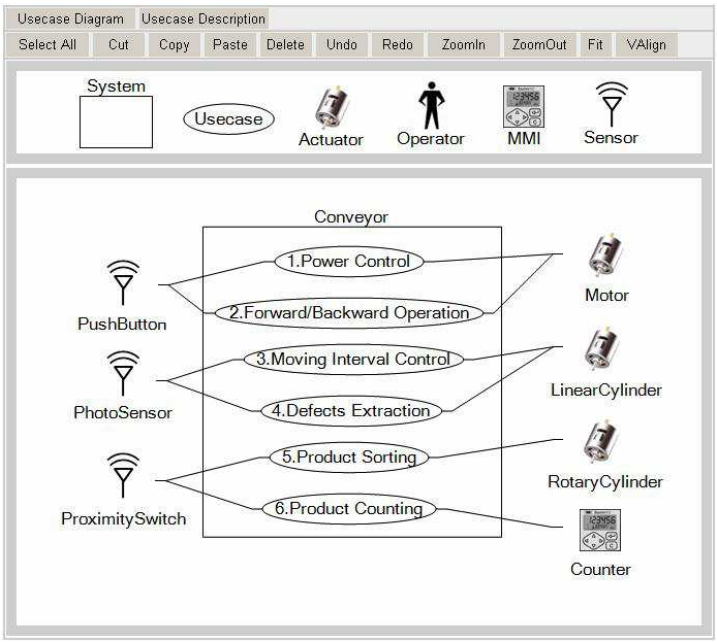

Fig. 2. Use case diagram of example prototype

The details of each use case are described in a use case description list. The use case description list includes the pre-/post-condition of a use case and interactions of a PLC with its actors such as sensors and actuators. For realizing a use case, related domain classes accomplish an allocated responsibility through the interactions among them. These related classes are identified in the structure model. And the system-level interactions in a use case description list are described in more detail in the interaction model.
Figure 2 and 3 shows a functional model for the example system in the form of use case diagram and use case description list. This example application prototype, as depicted in

Figure 4, is a kind of conveyor-based material handling system which identifies defective products according to their height, extracts defective products, and sorts good products according to their material property. It has 6 use cases for describing major functions from power control to product counting as depicted in Figure 2. Figure 3 shows the use case description of use case 4 (defects extraction) in Figure 2, and describes the high-level interactions between system (PLC) and its actors such as photo sensors and cylinders.

Usecase Description	
System : ConveyorSystem	Name : Defects extration
Scenario : Identifies defective products using 2 photo sensors. If defect product is identified, controller actuators extraction cylinder for the removal of product.	
Pre-condition : identification of defective products	
Post-condition : extraction of defective products	
Typical course or Events	
Actor	System
1. Product arrives 1.1 high level sensor is OFF and low level sensor is ON 1.2 high&low level sensors are all ON 1.3 high&low level sensor are all OFF 3. Extraction point sensor senses product 5. Extract defective product by forward stroke 6. Proximity switch senses good product	2. Identifies state of product 2.1 identifies good product 2.2/2.3 identifies defective product 4. Controller control cylinder according to the state of product 4.1 in case of defective products, sends forward stroke signal 4.2 initialize product status memory 7. Initialize product status memory

Fig. 3. Use case description list of example prototype

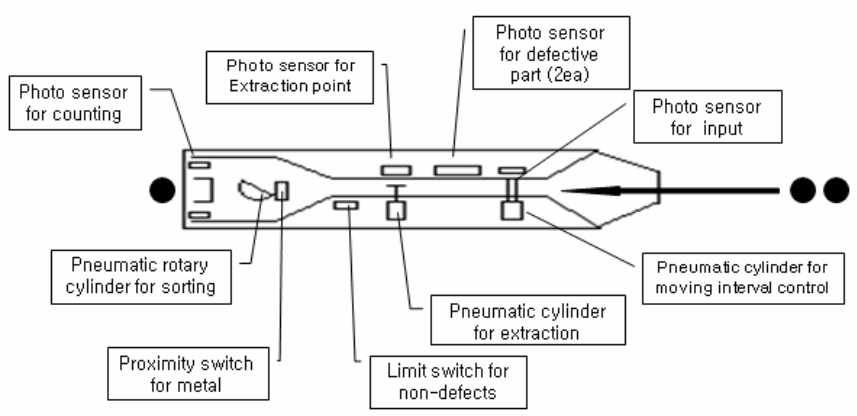

Fig. 4**.** Structure of example application prototype

A generic AMS is comprised of 4 parts: there is a 'plant' for manufacturing products. A plant is controlled by a 'controller' (PLC) which is managed by an 'operator' who monitors

plant through MMI. A 'work piece' flows through a plant. A plant is further decomposed into standard resource groups hierarchically.
Any standard resources can be classified using 3-level hierarchy of resource group-device group-standard device: A plant is composed of 'resource group' such as mechanical parts, sensor, actuator, and MMI. A resource group consists of 'device group'. For example, actuator resource group is composed of solenoid, relay, stepping motor, AC servo motor, and cylinder device group and so on. Sensor resource group is composed of photo sensor, proximity switch, rotary encoder, limit switch, ultrasonic sensor, counter, timer, and push button device group and so on. Finally, device group consists of 'standard devices' which can be acquired at the market.
To facilitate the modular design concept of modern AMS, the structure of AMS is modeled using an UML class diagram based on the proposed generic AMS structure. By referencing this generic AMS structure, FA engineers can derive the structure model of specific AMS reflecting special customer requirements easily. Figure 5 represents a static structure model of an example application prototype. Various kinds of device group class such as proximity switch and counter are inherited from generic resource group class such as sensor.

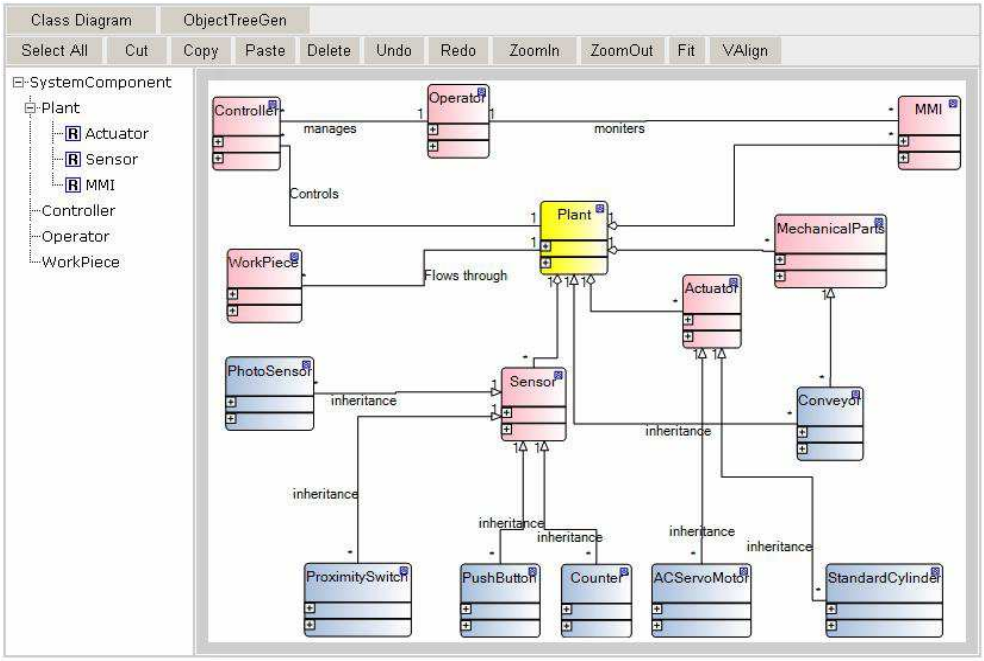

Fig. 5. Class diagram of example prototype

Since the real FA system is operated by the signal sending and receipt among manufacturing equipments such as PLC, sensors, and actuators, it is essential to describe the interactions of FA system components in detail for the robust design of device level control. This detail description of interactions is represented in the interaction model.
UML provides the activity diagram, state diagram, sequence diagram, and communication diagram as a modeling tool for dynamic system behaviors. Among these diagrams, the

activity diagram is most suitable for the control logic flow modeling because of following features: 1) it can describe the dynamic behaviors of plant with regard to device-level input/output events in sequential manner. 2) It can easily represent typical control logic flow routing types such as sequential, join, split, and iteration routing. The participating objects in the activity diagram are identified at the structure model.
In order to design and generate ladder logic, modification and extension of standard UML elements are required to reflect the specific features of ladder logic. First of all, it should be tested whether UML activity diagram is suitable for the description of control logic flow, especially for the ladder logic flow. The basic control flow at the ladder logic is sequence, split and join. Especially, three types of split and join control flow must be provided for ladder logic: OR-join, AND-join, AND-split. UML activity diagram can model basic control flows of ladder logic well.
Basically, ladder diagram is a combination of input contact, output coil and AND/OR/NOT logic. Since 'NOT' (normally closed) logic flow in the ladder logic cannot be represented directly in standard UML activity diagram, new two transition symbols for representing normally closed contact and negated coil are added as normal arcs with left-side vertical bar (called NOT-IN transition) or with right-side vertical bar (called NOT-OUT transition) as depicted in Figure 6. In the extended UML activity diagram, logic and time sequence flow from the top to the bottom of diagram.

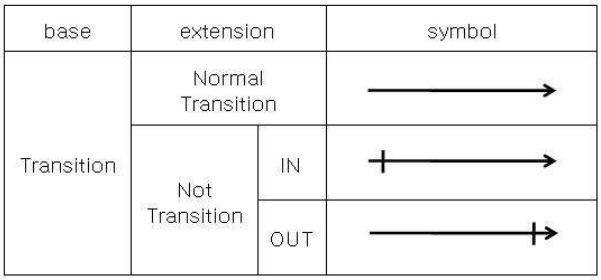

Fig. 6. Extensions of transitions in AD

Figure 7 represents the interaction model for the identification and extraction of defective parts according to the height of products at the example application prototype. (Refer the use case number 4 in Figure 2 and use case description in Figure 3)
The control logic of Figure 7 for defects extraction is as follows: 1) High_Memory:= (High_Sensor + High_Memory) * !Extract_Cyl, 2) Low_Memory:= (Low_Sensor + Low_Memory) * !Extract_Cyl * !OK_LimitSwitch, 3) Extract_Cyl:= {(High_Memory * Low_Memory) + (!High_Memory * !Low_Memory)} * Extract_Sensor where "!" means negation (NOT), '*' means conjunction (AND), and '+' means disjunction (OR).

4. O-O simulation for validating control logic

In this phase, O-O simulation model is constructed, and is executed for validating the designed control logic. When logic errors are found during the simulation execution, FA engineers correct logic errors and run the simulation model again. After validating control logic through simulation, FA engineers modify UML design model for reflecting the simulation result. In this way, the design-simulation cycle is done iteratively for error-free control logic.

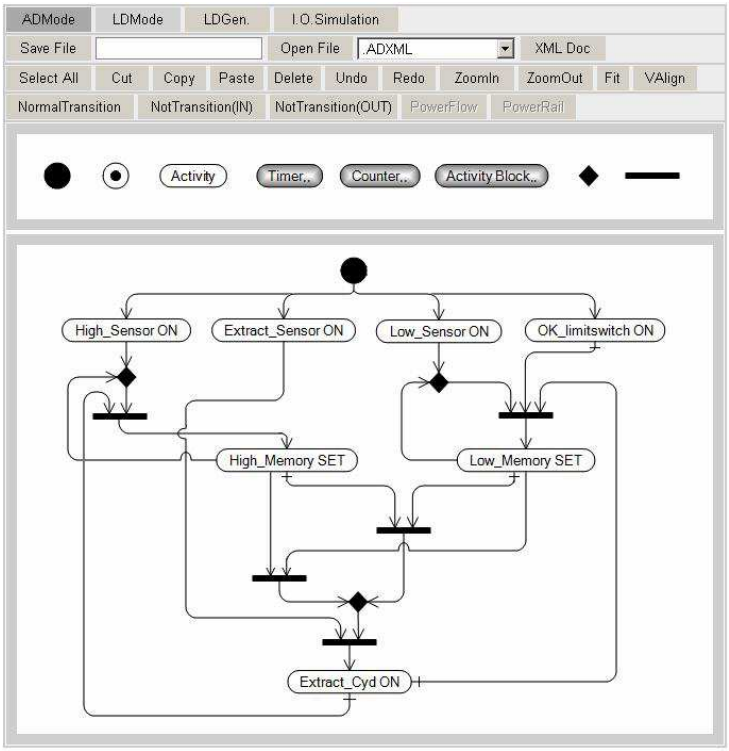

Fig. 7. Extended activity diagram for use case 4 in Figure 2 of example prototype

4.1 Construction of O-O simulation model

Based on the results of the O-O design model described in the Section 3, the O-O simulation model is constructed. The Unigraphics emPLANT software is used as an O-O simulation tool (Unigraphics, 2006).

First, for constructing an O-O simulation model, top-level functional requirements of automated manufacturing system are specified by using the use case diagram (Figure 2), and system-level interactions between the PLC and device actors (i.e., sensors and actuators) are identified by using a use case description list (Figure 3).

Second, AMS classes at the structure model (Figure 5) are mapped to emPLANT classes using the system hierarchy and association/inheritance relations among AMS classes identified in the class diagram. The mapping between generic AMS classes and emPLANT classes is summarized in Table 1.

Lastly, after determining the static system structure, control logic among system components is implemented for realizing each use case specified in the interaction model. The internal logic in the activity diagram is programmed in the simulation model by using SimTalk language of emPLANT software. For example, defects extraction of Figure 7 is executed by defect part identification and actuating extract cylinder. Detail control logic of this method is as follows: First, it inspects the product status according its height (a defective or good part). According to the inspection result, internal memories for high-level

and low-level detection are updated. If the product is defective and the sensor for extraction point is 'ON', the controller actuates an extraction cylinder The SimTalk code of this logic is described in Table 2.

Generic AMS class		emPLANT class
Controller		Frame/ Method
Workpiece		Entity
Plant	Sensor	SingleProc/ Line-sensor
	Actuator	SingleProc
	Mechanical parts	Line/ SingleProc/ Transporter
	MMI	Frame/Method

Table 1. Mapping between generic AMS classes and O-O simulation elements

```
.Models.PLC.extract_cyd_ON
{
is
do
if ((high_Sensor=1 or high_Memory=1) and (extract_Cyl=0))
then .models.conveyor_system.Plant.set_High_Memory;
end;
if ((low_Sensor=1 or low_Memory=1) and (extract_Cyd=0) and (OK_Limit_Switch=0))
then .models.conveyor_system.Plant.set_Low_Memory;
end;
if ((high_Memory=1 and low_Memory=1) or (high_Memory=0 and low_Memory=0)) and (extract_Sensor=1)
then .models.conveyor_system.Plant.set_Extract_Cyl;
end;
end;
}
```

Table 2. Example of SimTalk simulation code for Figure 7

4.2 Execution of O-O simulation model

The main characteristics of the O-O model is the easiness of a top-down modeling approach because extended new classes which share common properties can be created by inheriting the pre-defined classes, and a system can be decomposed into sub-systems hierarchically.

The O-O simulation model of an example application prototype has two-level hierarchy. The high-level model for example prototype consists of a controller (PLC), a plant, a source of products, a storage of defective products, and a storage of good products (upper right part of Figure 8). Furthermore, this prototype can be abstracted to 2 components (i.e., a controller and a plant). The low-level model, which is a base model of simulation execution, decomposes the high-level model into more detailed elements such as sensors, actuators,

and MMI (lower right part of Figure 8). After constructing a simulation model and preparing an experimental frame, a simulation model can be executed in which product flows are animated through the conveyor line.

In parallel with the animation of products flow, the proposed O-O simulation model can show the animation of PLC operations in response to the various events about product flows (Left part of Figure 8). When the sensing of a product by various sensors is signaled to the input port of a PLC (input 'ON' signal), a PLC executes corresponding control logic and sends a signal to the output port of a PLC (output 'ON' signal). The output 'ON' signal is transmitted to the actuator, so the actuator is enabled.

As depicted in Figure 8, during the simulation execution, the ON/OFF animation of the PLC input/output ports is displayed in parallel with the product flows. Input ports are located at the left side of a PLC, and output ports are located at the right side of a PLC. The 'ON' signal of input/output ports is displayed by a red color at the screen display.

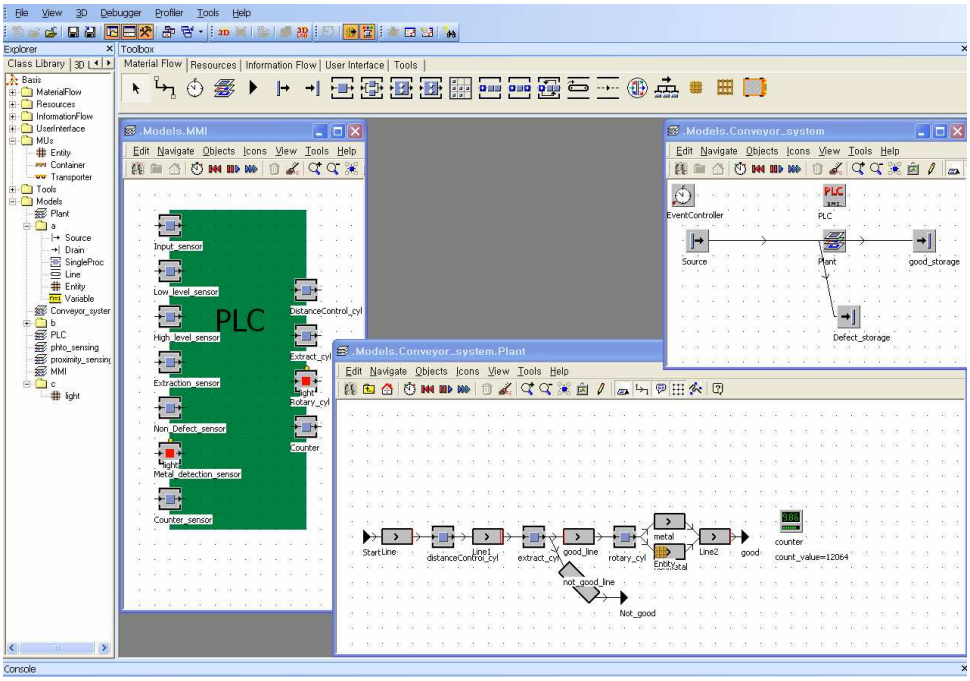

Fig. 8. O-O simulation model for example application prototype

Through the O-O simulation execution, PLC programmers can easily validate the internal logic of a PLC, and detect the logic errors at an earlier stage of the logic development by concurrent checking of product flows and PLC input/output port operations. Therefore, by adopting the proposed O-O simulation method, the validation of PLC control logic can be performed in parallel with the conventional performance evaluation.

5. Automatic generation of ladder code and its verification

The following two steps are conducted during the automatic generation phase: Firstly, ladder code is generated automatically using the interaction model result of design phase.

Secondly, generated ladder code is verified by input/output port-level simulation. In this phase, a software tool developed by research group including author is also used.

For the automatic generation of ladder logic, the mapping scheme of an UML activity diagram to a ladder diagram is established. IEC61131-3 standard ladder diagram have 5 major elements: contact, coil, power flow, power rail and function block (FB). Contact is further classified to normally open and normally closed contact. Coil is further classified to normal and negated coil. Power flow is further classified to vertical and horizontal power flow. Power rail is further classified to left and right power rail.

Elements of an activity diagram are classified to two types: an activity type and a transition type. Activity type is decomposed into start/stop activity, normal activity, special activity such as counter and timer, and block activity (Refer Figure 7). Transition type is decomposed into normal transition, NOT-IN transition for normally closed contact, NOT-OUT transition for negated coil, and logic flow transition. Logic flow transition is further decomposed into OR-join, AND-join and AND-split.

Figure 9 shows mapping scheme from an activity diagram to a ladder diagram. In order to store graphical activity diagrams and ladder diagrams in computer readable form, XML schema called AD-XML and LD-XML is devised for each diagram. In particular, LD-XML is an extension of PLCopen XML format (PLC Open, 2005).

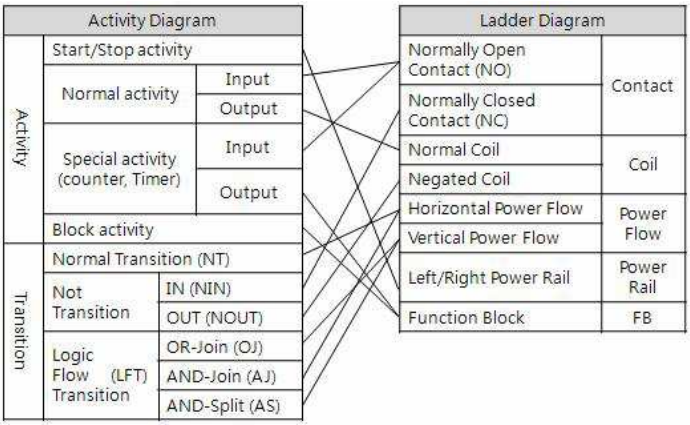

Fig. 9. Mapping scheme from AD to LD

After the activity diagram for specific control logic is stored in the form of AD-XML, AD-to-LD transformation procedure is conducted. Since basic ladder lung is a combination of input contact and output coil, an activity diagram is needed to be decomposed into several transformation units which having input(s) and output(s) corresponding to each ladder lung. This basic transformation unit is called IOU (Input Output Unit) which is a 1:1 exchangeable unit to ladder lung except start/stop activity. For example, the activity diagram depicted in Figure 10, which describes of power control logic (use case number 1 in Figure 2), has three IOUs. The control logic of Figure 10 is as follows: Conveyor_Motor: = (PowerON_ Button + Conveyor_Motor) * !PowerOFF_Button.

The transformation procedure is as follows: 1) After the creation of an activity diagram graphically, store it in the form of AD-XML. 2) Decompose an activity diagram into several input/output units called IOUs, and store it in the form of two-dimensional table called

IOU-Table. IOU-table has four columns named input activity, transition, output activity and IOU pattern type. Each row of IOU-Table becomes a part of ladder lung after the transformation process. 3) Determine the pattern type for each identified IOU. There are five IOU pattern types of activity diagram from the start/stop IOU type to the concatenation of logic flow transition IOU type. Generated IOU table for Figure 10 is shown at Table 3. 4) Finally, generate ladder lungs using IOU table and node connection information of AD-XML.

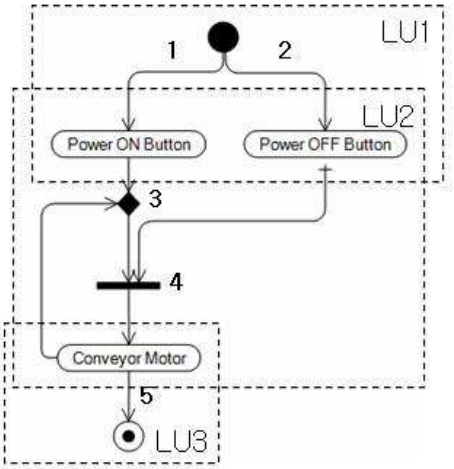

Fig. 10. IOU (Input Output Unit) decomposition

No.	Input Activity	Transition	Output Activity	Pattern
1	Start	1 : NT, 2 : NT	Power ON Button Power OFF Button	Type 1
2	Power ON Button Power OFF Button Conveyor Motor	3 : OJ, 4 : AJ	Conveyor Motor	Type 5
3	Conveyor Motor	5 : NT	Stop	Type 1

Table 3. IOU table for Figure 10 (use case 1-power control in Figure 2)

Figure 11 shows five IOU types and their corresponding LD patterns. IOU pattern type is classified to two types. One is simple type that is transformed to several basic ladder elements. The other is complex type that is a combination of simple types. Simple type is further classified to four types according to their corresponding lung structure: Type-1 (start/stop IOU), Type-2 (basic IOU), Type-3 (logic flow transition IOU: OR-join, AND-join, AND-split), and Type-4 (basic IOU with function block).

Since complex type is combination of several consecutive logic flow transitions, it has most sophisticated structure among 5 IOU types. Complex type is further classified to two types: Type 5-1 (join precedent) and Type 5-2 (split-precedent). Classification criteria is whether 'join' logic flow transition is precedent to other logic flow transitions or 'split' transition is precedent.

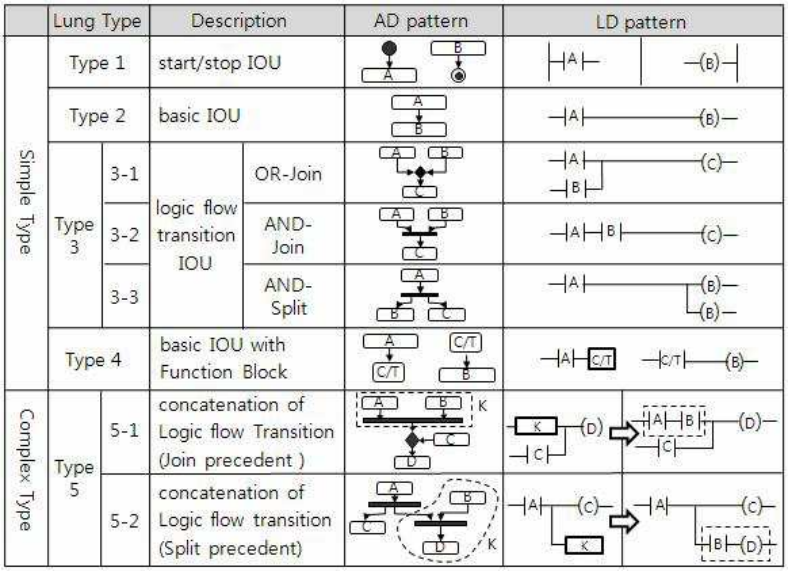

Fig. 11. Five IOU types

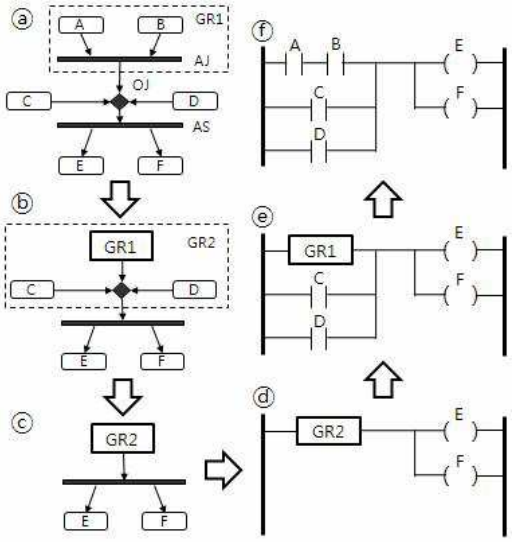

Fig. 12. Transformation procedure of join-precedent type 5-1

In order to transform the type-5 IOU to ladder pattern, hierarchical multi-step procedure is needed. The type-5 IOU is grouped hierarchically into several macro blocks for simplifying the consecutive control logic. A macro block is considered as a kind of block activity. Later, one macro block is transformed to one of five LD patterns. In other words, in order to simplify inputs for succeeding logic flow transition, firstly a macro block is built including precedent or succeeding logic flow transition. Later, a macro block is substituted by one of 5

ladder lung pattern. Fig. 12 shows the example of transformation procedure for the join-precedent type 5-1.
Ladder code is automatically generated based on the IOU table and node connection information of AD-XML. The generated ladder code is stored in the form of LD-XML, and is graphically displayed by reading LD-XML file as depicted in Figure 13. After ladder code is generated, it is necessary to verify the generated code. The simulation for code verification is conducted by input/output port level.
The ladder diagram in Figure 13 is generated from the control logic of activity diagram in Figure 7. As depicted in Figure 13, one can simulate the result of logic flow by closing or opening an input contact of specific lung, and monitoring the result of output coils and input contacts of other lungs.

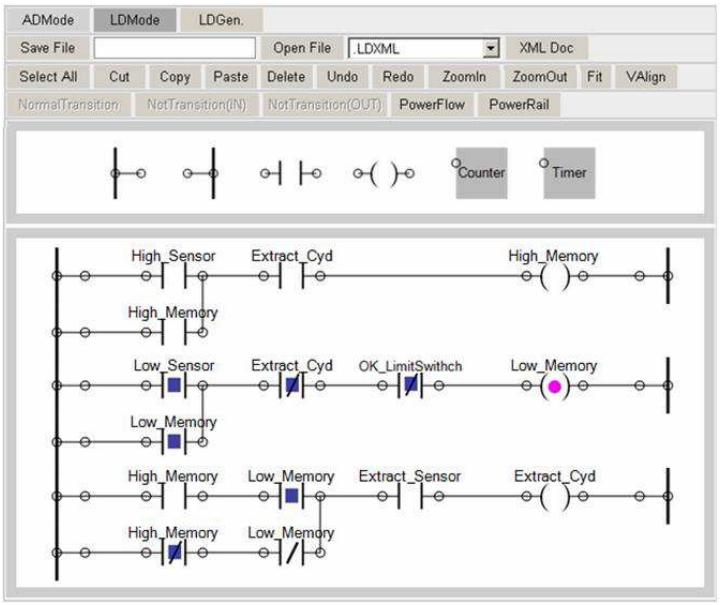

Fig. 13. Ladder code genration and port-level simulation of Figure 7

6. Conclusion

Currently, most enterprises do not adopt systematic development methodologies for ladder logic programming. As a result, ladder programs are error-prone and require time-consuming tasks to debug logic errors. In order to improve current PLC programming practices, this chapter proposes an integrated object-oriented ladder logic development framework in which control logic is designed, validated, generated automatically, and finally verified.
Proposed framework consists of three phases: First is the design phase. Second is the simulation phase. Third is the generation and verification phase. During the phase I, object-oriented design model is built, which consists of three sub-models: functional sub-model, structure sub-model and interaction sub-model. Based on the design result, O-O simulation model is constructed and executed for validating control logic during Phase II. After

correcting logic errors in Phase II, two steps are conducted during the phase III. Firstly, ladder code is generated automatically using the validated interaction model of design phase. Secondly, generated ladder code is verified by input/output port simulation.
A framework in this chapter facilitates the generation and modification of ladder code easily within a short time without considering complicated control behavior to deal with current trend of reconfigurable manufacturing systems. In addition, this framework serves as a helpful guide for systematic ladder code development life cycle.
As a future research, reverse transformation method from a ladder diagram to an activity diagram is needed for the accumulation of ladder logic design documents since design documents of control logic are not well prepared and stored in the shop floor.

7. References

Baresi L., Mauri M., Monti A., and Pezze M. (2000). PLCTools: design, formal validation, and code generation for programmable controllers, *Proceedings of 2000 IEEE Conference on Systems, Man and Cybernetics*, Nashville, USA

Bruccoleri M., and Diega S. N. (2003). An object-oriented approach for flexible manufacturing control systems analysis and design using the unified modeling language, *International Journal of Flexible Manufacturing System*, Vol.15, No.3, pp.195-216

Calvo I., Marcos M., Orive D., and Sarachaga I. (2002). Using object-oriented technologies in factory automation, *Proceedings of 2002 IECON Conference*, pp.2892-2897, Sevilla, Spain

Choi B.K., Han K.H., Park T.Y., (1996). Object-oriented graphical modeling of FMSs, *International Jouranl of Flexible Manufacturing System* , Vol.8, No.2, pp.159-182

Frey G. and Minas M. (2001). Internet-Based Development of logic controllers using signal interpreted Petri nets and IEC 61131, *Proceedings of the SCI 2001*, Vol.3, pp.297-302, Orlando, FL, USA

Hajarnavis V. and Young K. (2005). A comparison of sequential function charts and object modeling with PLC Programming, *Proceedings of American Control Conference*, pp.2034-2039

Han K. H. and Park J. W. (2007a). Development of object-oriented modeling tool for the design of industrial control logic, *Proceedings of the 5th International Conference on SERA*, pp.353-358, Busan, Korea

Han K. H. , Park J. W. and Choi Y. (2007b). Object-oriented modeling and simulation for the validation of industrial control logic, *Proceedings of the 37th international conference on CIE*, pp. 2377-2384, Alexsandria, Egypt

Jack H. (2007). Automating manufacturing systems with PLCs. http://claymore.engineer.gvsu.edu/~jackh/books.html

Lee G. B., Zandong H. and Lee J. S. (2004). Automatic generation of ladder diagram with control Petri net, *Journal of Intelligent Manufacturing*, Vol.15, No.2, pp.245-252

Liu J. and Darabi H. (2002). Ladder Logic Implementation of Ramadge-Wonham supervisory controller, *Proceedings of Sixth International Workshop on Discrete Event Systems,* pp.383-389

Maffezzoni C., Ferrarini L., and Carpanzano E. (1999). Object-oriented models for advanced automation engineering, *Control Engineering Practice*, Vol.7, No.8, pp.957-968

Manesis S. and Akantziotis K. (2005). Automated synthesis of ladder automation circuits based on state diagrams, *Advances in Engineering Software*, Vol.36, No.4, pp.225-233

Park T.Y., Han K.H., Choi B.K., (1997). An object-oriented modeling framework for automated manufacturing systems, *International Journal of Computer Integrated Manufacturing*, Vol.10, No.5, pp.324-343.

Peng S. S. and Zhou M. C. (2004). Ladder diagram and Petri net based discrete event control design methods, *IEEE Transactions on Systems, Man and Cybernetics-Part C.*, Vol.34, No.4, pp.523-531

PLC Open (2005). XML formats for IEC 61131-3, http://www.plcopen.org

Sacha K. (2005). Automatic code generation for PLC controllers, *LNCS 3688*, pp.303-316

Spath D., and Osmers U. (1996). Virtual reality- an approach to improve the generation of fault free software for programmable logic controllers, *Proceedings of IEEE International Conference on ECCS,* pp.43-46, Montreal, Canada

Taholakian A. and Hales W. M. M. (1997). PN <-> PLC: a Methodology for designing, simulating and coding PLC based control systems using Petri nets, *International Journal of Production Research*, Vol.35, No.6, pp.1743-1762

Unigraphics (2006), emPlant, www.ugs.com/products/tecnomatix/plant_design/em_plant.shtml.

Young K. W., Piggin R., and Rachitrangsan P. (2001). An object-oriented approach to an agile manufacturing control system design, *International Journal of Advanced Manufacturing Technology*, Vol.17, No.11, pp.850-859

Zhou M. C. and Venkatesh K (1999). Modeling, simulation and control of flexible manufacturing systems, *World scientific publishing*, Farrer Road, Singapore

Programmable Logic Controller

Edited by Luiz Affonso Guedes

ISBN 978-953-7619-63-3
Hard cover, 170 pages
Publisher InTech
Published online 01, January, 2010
Published in print edition January, 2010

Despite the great technological advancement experienced in recent years, Programmable Logic Controllers (PLC) are still used in many applications from the real world and still play a central role in infrastructure of industrial automation. PLC operate in the factory-floor level and are responsible typically for implementing logical control, regulatory control strategies, such as PID and fuzzy-based algorithms, and safety logics. Usually PLC are interconnected with the supervision level through communication network, such as Ethernet networks, in order to work in an integrated form. In this context, this book was written by professionals that work and research in automation area and it has two major objectives. The first objective is present some advances in methodologies and techniques for development of industrial programs based on PLC. The second objective is present some PLC-based real applications from various areas such as manufacturing system, robotics, power system, communication system, and education.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Kwan Hee Han (2010). Object-Oriented Modeling, Simulation and Automatic Generation of PLC Ladder Logic, Programmable Logic Controller, Luiz Affonso Guedes (Ed.), ISBN: 978-953-7619-63-3, InTech, Available from: http://www.intechopen.com/books/programmable-logic-controller/object-oriented-modeling-simulation-and-automatic-generation-of-plc-ladder-logic

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821