Visual Validation Of PLC Programs

Min.S Ko
Department of Industrial Engineering
Ajou University
Suwon, Korea

Sang C. Park
Department of Industrial Engineering
Ajou University
Suwon, Korea

Gi Nam. Wang
Department of Industrial Engineering
Ajou University
Suwon, Korea

KEYWORDS
PLC verification, plant model, virtual device model, virtual factory simulation

ABSTRACT

Proposed in this paper is the architecture of a PLC programming environment that enables a visual verification of PLC programs. The proposed architecture integrates a PLC program with a corresponding plant model, so that users can intuitively verify the PLC program in a 3D graphic environment. The plant model includes all manufacturing devices of a production system as well as corresponding device programs to perform their tasks in the production system, and a PLC program contains the control logic for the plant model. For the implementation of the proposed PLC programming environment, it is essential to develop an efficient methodology to construct a virtual device model as well as a virtual plant model. The proposed PLC programming environment provides an efficient construction method for a plant model based on the DEVS (Discrete Event Systems Specifications) formalism, which supports the specification of discrete event models in a hierarchical, modular manner.

1.INTRODUCTION

Generally, industrial production lines are dynamic systems whose states change according to the occurrence of various events, thus exhibiting the characteristics of a discrete event system. If manufacturers are to remain competitive in a continuously changing marketplace, they must not only continue to improve their products, but also strive to improve production systems continuously (B.K Choi et al. 2000). Thus, an efficient prototyping environment for production systems is crucial. A modern production line is a highly integrated system composed of automated workstations such as robots with tool-changing capabilities, a hardware handling system and storage system, and a computer control system that controls the operations of the entire system. The implementation of a production line requires much investment, and decisions at the design stage have to be made very carefully to ensure that a highly automated manufacturing system will successfully achieve the intended benefits.

Simulation is an essential tool in the design and analysis of complex systems that cannot be easily described by analytical or mathematical models (Klingstam. 1999; Al-Ahmari 1999.). It is useful for calculating utilization statistics, finding bottlenecks, pointing out scheduling errors and even for creating manufacturing schedules. Traditionally, various simulation languages, including ARENA® and AutoMod®, are used for the simulation of manufacturing systems. Those simulation languages have been widely accepted both in industry and in academia; however, they remain as analysis tools for the rough design stage of a production line, because their simulation models are not realistic enough to be utilized for a detailed design or for implementation purposes. For example, real production lines are usually controlled by PLC (Programmable Logic Controller) programs (Rukkan A 1997), as shown in Figure 1, but conventional simulation languages roughly describe the control logic with independent entity flows (job flows) between processes.

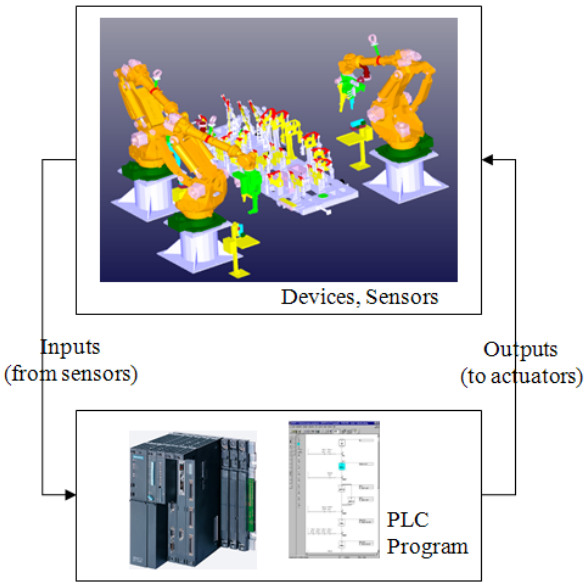

Figure 1: Production system controlled by a PLC program.

For a detailed design (virtual prototyping) of a production line, it is necessary to create a much more detailed simulation model that can forecast not only the production capability of the system but also the physical validity and efficiency of co-working machines and control programs. As shown in Figure 1, various

Proceedings 22nd European Conference on Modelling and Simulation ©ECMS Loucas S. Louca, Yiorgos Chrysanthou, Zuzana Oplatková, Khalid Al-Begain (Editors)
ISBN: 978-0-9553018-5-8 / ISBN: 978-0-9553018-6-5 (CD)

machines that operate simultaneously in an industrial manufacturing system are usually controlled by PLCs, currently the most suitable and widely employed industrial control technology (Chuang. 1999; Manesis. 2005; Rullan. 1997; Jang. 1997). A PLC (Programmable Logic Controller) emulates the behavior of an electric ladder diagram. As they are sequential machines, to emulate the workings of parallel circuits that respond instantaneously, PLCs use an input/output image table and a scanning cycle. When a program is being run in a PLC it is continuously executing a scanning cycle. The program scan solves the Boolean logic related to the information in the input table with that in output and internal relay tables. In addition, the information in the output and internal relay tables is updated during the program scan. In a PLC, this Boolean logic is typically represented using a graphical language known as a ladder diagram (Rullan. 1997).

Various software tools have been developed for the modeling and verification of PLC-based systems via the use of timed automata, such as UPPAAL2k, KRONOS, Supremica and HyTech, mainly for programs written in a statement list language also termed Boolean (Manesis. 2005). Those software tools verify PLC programs to a certain extent; however, they remain limited. In particular, it is not easy for users to determine whether the PLC programs actually achieve the intended control objectives.

The objective of this paper is to propose the architecture of a PLC programming environment that enables the visual validation of a PLC program. The proposed PLC programming environment employs a virtual plant model consisting of virtual devices, so that users can easily verify the PLC program. The overall structure of the paper is as follows. Section 2 illustrates the architecture of the proposed PLC programming environment, while Section 3 describes an efficient construction methodology for a plant model, which can be synchronized with a PLC program. Section 4 shows an example and illustrations. Finally, concluding remarks are given in Section 5.

2. VISUAL VALIDATION OF PLC PROGRAMS

To design the architecture of the PLC programming environment, it is important to understand the basic procedure used to construct a PLC program (ladder diagram). (Chuang et al. 1999) proposed a procedure for the development of an industrial automated production system that consists of nine steps. They are: 1) Define the process to be controlled; 2) Make a sketch of the process operation; 3) Create a written sequence listing of the process step by step; 4) On the sketch, add the sensors needed to carry out the control sequence; 5) Add the manual controls needed for the process-setup or for operational checks; 6) Consider the safety of the operating personnel and make additions and adjustments as needed; 7) Add the master stop switches required for a safe shutdown; 8) Create a ladder logic diagram that will be used as a basis for the PLC program; and 9) Consider the possible points where the process-sequence may go astray. The most time-consuming task for the control logic designers is the 8-th step, which is usually done by the repetitive method of 'Code writing, testing and debugging' until the control objectives are achieved (Manesis. 2005). The bottleneck of the 8-th step is that the conventional PLC programming environments are not especially intuitive, particularly for the testing and debugging of a PLC program, as they show only the status of a PLC without providing any links to the target system (production line). For the validation of a PLC program, engineers need to imagine the state changes of a production line from the input and output ports of a PLC. That is the reason conventional PLC programming environments are often inefficient and prone to human error. As the configurations of production lines and their control programs become more complicated, there is a strong need for a more intuitive PLC programming environment. It is hoped that this paper will take positive steps in this direction.

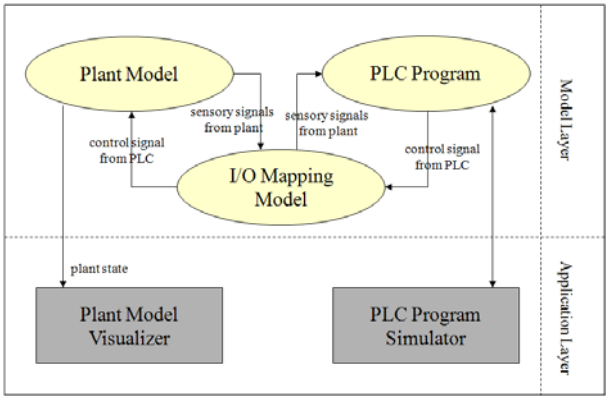

Figure 2: The proposed PLC programming environment

Figure 2 shows the architecture of the proposed PLC programming environment. It consists of two layers, a model layer and an application layer. The model layer has three models, a plant model (virtual factory model), a PLC program (control model) and an I/O mapping model. The plant model includes all manufacturing devices of the production system as well as the corresponding device programs to perform their tasks in the production system, and the PLC program contains the control logic for the plant model. For the integration of the plant model and the PLC program, it is necessary to define the mapping between the plant model and the PLC program, which is described by the I/O mapping model. The application layer simultaneously provides two interfaces to users. The 'PLC simulator' performs the simulation of the control program, and the 'plant model visualizer' shows the corresponding plant model (3D graphic models) reflecting the changing states of the production system during the PLC simulation. Thus, it becomes much easier for users to verify the PLC program though the plant model visualizer.

Among the three models of the model layer, the plant model plays a key role in the proposed PLC programming environment. As mentioned earlier, the

plant model should contain all devices as well as the device control programs. Thus, it can be considered as a 'virtual factory model', a model executing manufacturing processes in computers as well as the real world (M. Onosato. 1993; K Iwata. 1995; L. Ye. 1997). To implement a virtual factory, it is necessary to construct digital models for all the physical and logical elements (entities and activities) of a real factory.

The plant model consists of manufacturing devices with their positions in the layout. To represent such a manufacturing device, this paper employs the concept of a virtual device: a digital model imitating the physical and logical aspects of a real device. A virtual device needs to maintain its relationships with other devices or PLC programs as well as the inherent attributes of the device, such as the kinematics and geometric shape. To do so, a virtual device is split into two parts; a shell and a core. The shell part can adapt to the different configurations of production systems, and the core part undertakes the inherent properties of the device. The concept of a virtual device is shown in Figure 3.

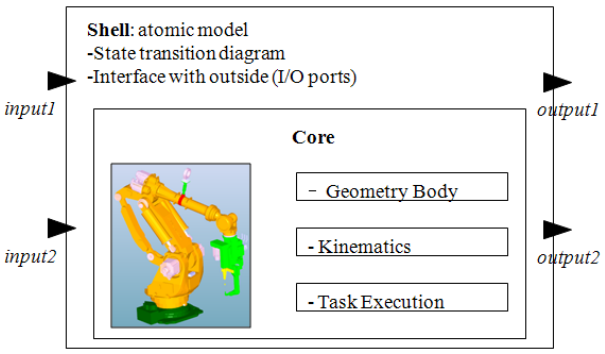

Figure 3: Virtual device model

The reusability of a virtual device is very important, as a virtual device can be used for many different configurations of production systems. Without the effective reusability characteristic of a virtual device, it is more common to create new components from scratch than to search for useful elements in other systems. For the reusability of a virtual device, it is essential for the shell part of a virtual device to be flexible enough to be compatible with any configuration of a production system. To build such a shell part, this paper employs Zeigler's DEVS (Discrete Event Systems Specifications) formalism (B.P.Zeigler. 1984; T.G.Kim. 1994), which supports the specification of discrete event models in a hierarchical, modular manner. The semantics of the formalism are highly compatible with object-oriented specifications for simulation models. Within the DEVS formalism, one must specify two types of sub-models: 1) the atomic model, the basic models from which larger models are built, and 2) the coupled model, how atomic models are connected in a hierarchical manner. Formally, an atomic model ***M*** is specified by a 7-tuple:

$$M =< X, S, Y, \delta_{\text{int}}, \delta_{ext}, \lambda, t_a > \quad (1)$$

X: input events set;
S: sequential states set ;
Y: output events set;
δ_{int} : ***S*** → ***S***: internal transition function;
δ_{ext} : ***Q*** * ***X*** → ***S***: external transition function
Q = {(*s*,*e*)| s ∈ ***S***, 0 ≤*e*≤ $t_a(s)$}: total state of ***M***;
λ : ***S*** → ***Y***: output function;
t_a : ***S*** → ***Real***: time advance function.

The four elements in the 7-tuple, namely $\delta_{\text{int}}, \delta_{ext}, \lambda$ and t_a, are called the characteristic functions of an atomic model. The second form of the model, termed a coupled model, shows a method for coupling several component models together to form a new model. Formally, a coupled model ***DN*** is defined as:

$$DN =< X, Y, M, EIC, EOC, IC, SELECT > \quad (2)$$

X: input events set;
Y: output events set;
M: set of all component models in DEVS;
EIC ⊆ ***DN.IN*** * ***M.IN***: external input coupling relation;
EOC ⊆ ***M.OUT*** * ***DN.OUT***: external output coupling relation;
IC ⊆ ***M.OUT*** * ***M.IN***: internal coupling relation;
SELECT: 2^{M} –Ø → ***M***: tie-breaking selector,

where the extensions .IN and .OUT represent the input port set and the output port set of the respective DEVS models. For the implementation of the plant model, the shell part of a virtual device is represented as an atomic model, and the entire plant model is represented as a coupled model, including the atomic models (virtual devices) and the coupling relationships between them. The detail specifications for the plant model are addressed in the following section.

3. PLANT MODEL CONSTRUCTION

The objective of the proposed PLC programming environment is to provide an intuitive PLC programming and verification environment by connecting the plant model to the PLC program. To achieve this objective, it is essential to develop an efficient construction procedure of a plant model. Figure 4 shows the interactions among three models of the PLC programming environment. The three models are a plant model, an I/O mapping model, and a PLC program. The plant model is controlled by the PLC program through the I/O mapping model.

Given that a plant model consists of virtual devices, the construction method of a virtual device is described

before explaining the construction method of a plant model. As explained earlier, a virtual device consists of a shell part and a core part.

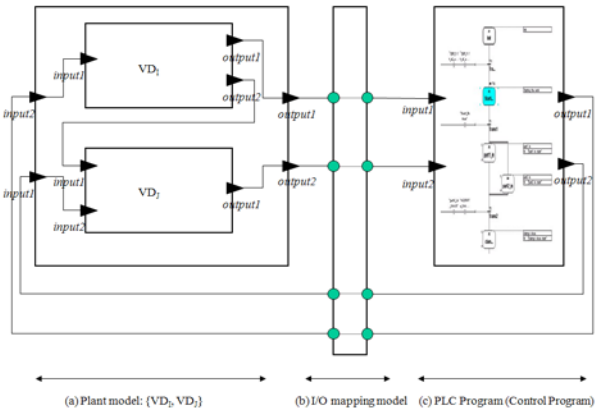

Figure 4: Interactions among three models of the PLC programming environment

The shell part, enclosing the core part (the inherent properties of a device, such as kinematics, geometric shape and the execution of device-level commands), should allow a virtual device model to adapt to different plant configurations. This part is modeled as an atomic model of the DEVS formalism, which is a timed-FSA (finite state automata). To define the shell part of a virtual device, first it is necessary to identify the set of tasks that are assigned to the device. The activation of each task is normally triggered by an external signal from either the PLC program or other virtual devices. Once the set of tasks is identified for a virtual device, it is then possible to extract the state transition diagram, which defines an atomic model of the DEVS formalism. Figure 4-(a) shows a simple example of an AGV (Automatic Guided Vehicle) with two tasks, *T1* (movement from p1 to p2) and *T2* (movement from p2 to p1). As the two tasks should be triggered by external events, the shell part of the AGV must have two input ports, termed here as *Signal_1* and *Signal_2*, as shown in Figure 4-(b). From the set of tasks, it is possible to instantiate the state transition diagram automatically. For this example, there are four states, *P1*, *DoT1*, *P2* and *DoT2*. While *P1* and *P2* take external events from the input ports (*Signal_1, Signal_2*) for state transitions, *DoT1* and *DoT2* take internal events that are the end events of the two tasks (*T1* and *T2*). The DEVS atomic model of the virtual device, corresponding to the AGV, can be described as follows:

Shell of a virtual device:

$$M = < X, S, Y, \delta_{\text{int}}, \delta_{ext}, \lambda, t_a > \quad (3)$$

$\boldsymbol{X}$ = {*Signal_1, Signal_2*}
$\boldsymbol{S}$= {*P1, DoT1, P2, DoT2*}
$\boldsymbol{Y}$= {*T1Done, T2Done*}

$\delta_{\text{int}}(DoT1) = P2$

$\delta_{\text{int}}(DoT2) = P1$

$\delta_{ext}(P1, Signal\_1) = DoT1$

$\delta_{ext}(P2, Signal\_2) = DoT2$

$\lambda(DoT1) = T1Done$

$\lambda(DoT2) = T2Done$

$t_a(DoT1) = \text{Time\_1}$

$t_a(DoT2) = \text{Time\_2}.$

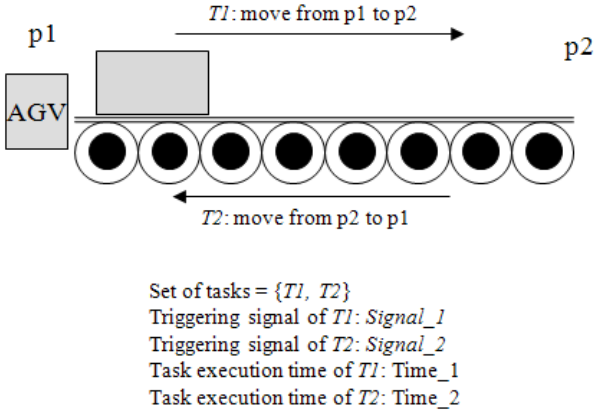

Figure 5-(a): Task identification of a device

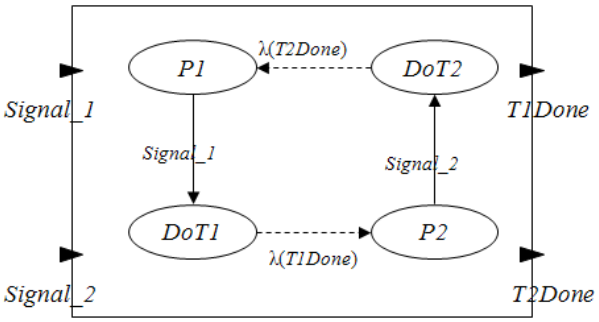

Figure 5-(b): State transition diagram

Once virtual device models are constructed, a plant model can be defined by combining the virtual devices. While virtual devices are described as atomic models, the entire plant model is modeled as a coupled model, including those atomic models and coupling relationships between them. Figure 4-(a) shows a simple example of a plant model including two virtual devices, VD_I and VD_J. The DEVS couple model of the plant model, shown in Figure 4-(a), can be described as follows:

Plant Model

$$DN = < X, Y, M, EIC, EOC, IC, SELECT > \quad (4)$$

$\boldsymbol{X}$ = {*input1, input2*}, $\boldsymbol{Y}$ = {*output1, output2*}
$\boldsymbol{M}$: = {VD_I, VD_J}
EIC= {(*DN.input1** VD_J.*input2*), (*DN.input2** VD_I.*input1*)}
EOC = {(VD_I.*output1*, *DN.output1*), (VD_J.*output1***DN.output2*)}
IC= {(VD_I.*output2** VD_J.*input1*)}
SELECT: $2^M - \emptyset \rightarrow \boldsymbol{M}$: tie-breaking selector.

The proposed methodology for the construction

of a plant model has two major benefits. The first is the reusability of a virtual device model, signifying that the structure of a virtual device model achieves independence from the configurations of a production system. The second benefit is the intuitiveness in defining the state transition diagram of the virtual device model. Users with only a passing knowledge of discrete event system modeling can easily define a virtual device model simply by identifying the set of tasks.

4. EXAMPLES & ILLUSTRATIONS

The prototype of the proposed PLC programming environment was implemented and tested with several examples. The C++ language in a Visual Studio environment was used, with OpenGL for the graphical rendering. An extrusion system, shown in Figure 6, consists of four sub-systems; 1) a feeder system, 2) a ram-die system, 3) a start button and 4) a stop button. Extrusion is a compression process in which the work metal is forced to flow through a die opening to produce a desired cross-sectional shape. The process can be likened to squeezing toothpaste out of a toothpaste tube (Groover. 2006). The control logic of the extrusion system is as follows:

(1) If the '*I-start*' signal is triggered by a user when the system is idle, then the PLC sends out a signal to the input port of the feeder system, '*I-pick*' = TRUE.
(2) If the '*I-pick*' becomes TRUE, then the feeder system starts the job for picking a billet. The output port of the feeder system '*O-picking'* becomes TRUE until the end of the job. When the picking is finished, '*O-picking'* becomes FALSE.
(3) If the job of picking a billet is finished ('*O-picking'* = FALSE), then the PLC sends out two signals, '*I-pick'* = FALSE and '*I-extrude'* = TRUE.
(4) If the port *'I-extrude'* becomes TRUE, then the extrusion system begins to compress the work metal (billet) to make it flow through a die opening. The output port of the extrusion system '*O-extruding'* becomes TRUE until the end of the extrusion. When the extrusion is finished, '*O-extruding'* becomes FALSE.
(5) When the extrusion of a billet is finished ('*O-extruding'* = FALSE), the PLC then sends out a signal, '*I-extrude'* == FALSE, to make the system state idle.
(6) Whenever the '*I-stop'* signal is triggered by a user, then all activities of the system should be stopped.

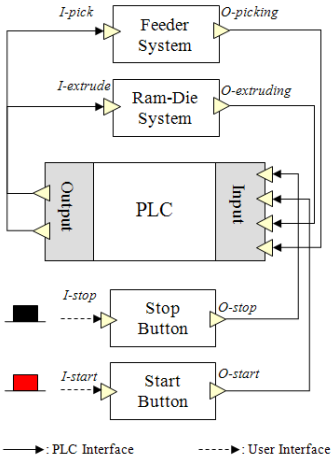

Figure 6: Configuration of an extrusion system

The control logic of the extrusion system should be converted into a PLC program for the implementation. Figure 7 shows the PLC program, written in the format of the SFC (Sequential Function Chart), which controls the extrusion system shown in Figure 6. The boxes along the vertical line (S1, S2, S3, S4 and S5) represent the states of the extrusion system, and the thick line segments between states (T1, T2, T3, T4, and T5) are transitions. Each transition has a ladder diagram on its left side that represents its firing condition. On the right side of the state boxes, there are additional boxes connected with a dotted line. Those boxes are the execution commands (R: reset, S: set) issued by the PLC that control the extrusion system. For the validation of the control logic, it is necessary to have a plant model enabling the visual validation of the PLC program. The plant model is shown in Figure 8, and it includes all devices (the feeder system, ram-die system, start button and stop button) of the production system as well as the corresponding device programs to perform their tasks in the production system.

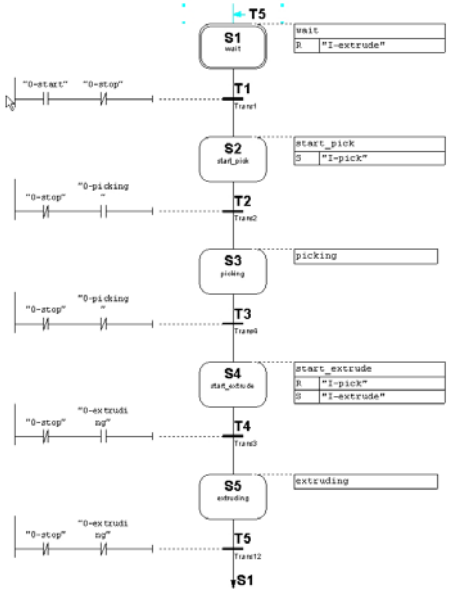

Figure 7: PLC program that controls the extrusion system

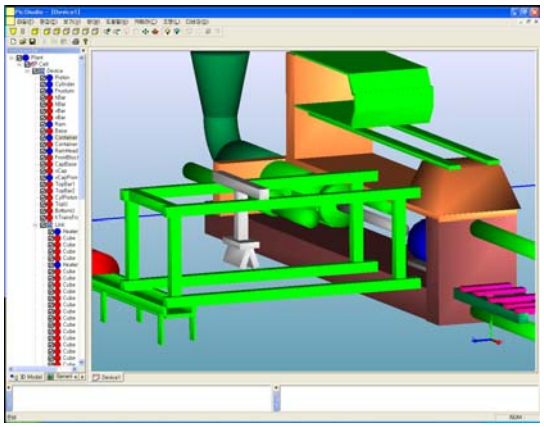

Figure 8: Plant model of the extrusion system

5. DISCUSSION AND CONCLUSIONS

This paper proposes the architecture of a PLC programming environment that enables a visual verification of a PLC program by synchronizing a PLC program with a corresponding virtual plant model. The model layer of the proposed architecture consists of three models: a plant model (virtual factory model), a PLC program (control model) and an I/O mapping model. The plant model includes all manufacturing devices of the production system, and the PLC program contains the control logic for the plant model. The I/O mapping model functions as a communication link between these two models. As the plant model plays a key role in the proposed PLC programming environment, it is essential to develop a practical methodology in the construction of a virtual device model as well as a virtual plant model. To do so, this paper addresses an efficient construction method of a plant model based on the DEVS (Discrete Event Systems Specifications) formalism, which supports the specification of discrete event models in a hierarchical, modular manner. The proposed methodology for the construction of a plant model has two major benefits. The first is the reusability of a virtual device model, signifying that the structure of the virtual device model achieves independence from the configurations of a production system. The second benefit is intuitively the defining of the state transition diagram of a virtual device model. It is not necessary for users to have in-depth knowledge of discrete event system modeling, as they simply have to identify a set of tasks in order to define a virtual device model.

REFERENCES

Al-Ahmari AMA, Ridgway K. 1999. "An integrated modeling method to support manufacturing system analysis and design." Computers in Industry. No.38:225-238.

Anglani A, Grieco A, Pacella M. 2002. "Tolio T. Object-oriented modeling and simulation of flexible manufacturing system: a rule-based procedure.", Simulation Modeling Practice and Theory, No.10:209-234.

B.K. Choi, B.H. Kim. 2000. "New trends in CIM: Virtual manufacturing systems for next generation manufacturing", *Current Advances in Mechanical Design and Production Seventh Cairo University Int. MDP Conf.*, Cairo, February 15-17, 425-436

B. P. Zeigler. 1984. "*Multifacetted modeling and discrete event simulation.*", Academic Press, Orland .

Chuang C P, Lan X, Chen J C. 1999. "A systematic procedure for designing state combination circuits in PLCs.", Journal of Industrial Technology, No.15(3):2-5.

Groover M P. 2006. "*Fundamentals of modern manufacturing.*", Wiley.

Jang J, Koo P H, Nof S Y. 1997. "Application of design and control tools in a multirobot cell.", Computers and Industrial Engineering, No.32:89-100.

K. Iwata, M. Onosato, K. Teramoto, S. Osaki., 1995. "A modeling and simulation architecture for virtual manufacturing systems.", CIRP, No. 44(1):399-402.

Klingstam P, Gullander P. 1999. "Overview of simulation tools for computer-aided production engineering.", Computers in Industry, No.38:173-186.

L. Ye, F. Lin., 1997. "Virtual system simulation – A step beyond the conventional simulation", *22nd Int. Conf. on Computer and Industrial Engineering*, 304-306.

Manesis S, Akantziotis K. 2005. "Automated synthesis of ladder automation circuits based on state-diagrams.", No.36:225-233.

M. Onosato, K. Iwata., 1993. "Development of a virtual manufacturing system by integrating product models and factory models.", CIRP, No.42(1):475-478.

Rullan A. 1997. "Programmable logic controllers versus personal computers for process control.", Computers and Industrial Engineering, No.33:421-424.

T. G. Kim., 1994, "*DEVSIM++ User's Manual*", Department of Electrical Engineering, KAIST, Korea, 1994.

AUTHOR BIOGRAPHIES

MIN SUK. KO was born in Korea, Je-Ju Island and went to the Ajou University of Suwon, where he studied Industrial Information System Engineering. He worked for a couple of years for the Digital Manufacturing Lab in 2007 to the University of Ajou where he is graduated student. His e-mail address is : sebaminsuk11@ajou.ac.kr and his Web-Page can be found at : http://udmtek.co.kr.

SANG CHUL. PARK. Is a professor in the Department of Industrial and Information System Engineering at Ajou University. Before joining Ajou, he worked for Daimler- Chrysler Corp. and CubicTek Co., developing commercial and in-house CAD/CAM/CAPP software systems. His research interests include geometric engineering knowledge management, and discrete event system simulation. His e-mail address is : scpark@ajou.ac.kr and his Web-Page can be found at : http://udmtek.co.kr.

GI NAM. WANG Is a professor in the Department of Industrial and Information System Engineering at Ajou University. His e-mail address is : gnwang@ajou.ac.kr and His Web-Page can be found at: http://udmtek.co.kr.