Proceedings of the 2008 Winter Simulation Conference
S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds.

PLCStudio: SIMULATION BASED PLC CODE VERIFICATION

Sang C. Park, Chang Mok Park, and Gi-Nam Wang

UDMTEk 113 Ho Sanhakwon, Ajou University,
Suwon, 442-739, South Korea

Jongeun Kwak, and Sungjoo Yeo

Dept. of Industrial and Information Systems
Engineering, Ajou University
Suwon 442-739, South Korea

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 [10]. 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 [5, 6]. 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 [14]. 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 [3], as shown in Fig. 1, but conventional simulation languages roughly describe the control logic with independent entity flows (job flows) between processes.

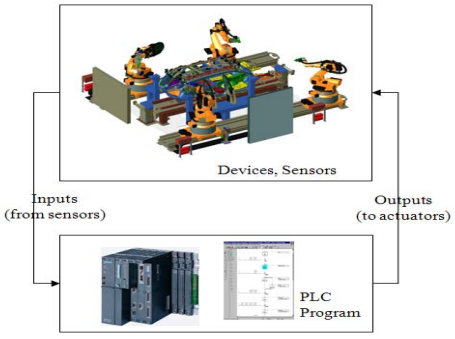

Figure 1: Production system controlled by a PLC program

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 Fig. 1, various machines that operate simultaneously in an industrial manufacturing system are usually controlled by PLCs, currently the most suitable and widely employed industrial control technology [1-4]. 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 [3].

Previous approaches on PLC programs can be categorized into two groups; 1) Verification of a given PLC program [18, 19], and 2) Generation of a dependable PLC program [15-17]. In the first group, various software tools have been developed for the 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 [2]. Those software tools verify PLC programs to a certain extent; however, they remain limited. Since they are mainly focusing on the checking of theoretical attributes (safety, liveness, and reachability), it is not easy for users to determine whether the PLC programs actually achieve the intended control objectives. In the second group, many researchers have focused on the automatic generation of PLC programs from various formalisms including state diagrams, Petri nets and IDEF0. Those formalisms can help the design process of control logics, however, it is still difficult to find hidden errors which are the most difficult part of the verification of a control program. To cope with the problem, we need a more transparent PLC programming environment helping users to recognize hidden errors.

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 Section3 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. [1] 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 [2]. 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.

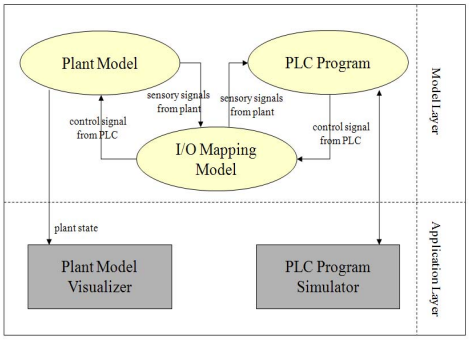

Figure 2: The proposed PLC programming environment

Fig. 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 through 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 [11-13]. 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 Fig. 3.

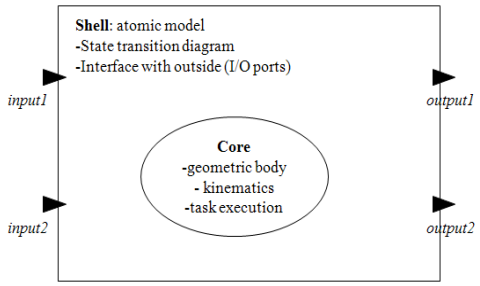

Figure 3: Virtual device model

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 [7, 8], 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:

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

X: input events set;
S: sequential states set ;
Y: output events set;
δ_{int} : $S \rightarrow S$: internal transition function;
δ_{ext} : $Q * X \rightarrow S$: external transition function
$Q = \{(s,e) | s \in S, 0 \le e \le t_a(s)\}$: total state of *M*;
λ : $S \rightarrow Y$: output function;
t_a : $S \rightarrow$ *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 couple model ***DN*** is defined as:

$DN =< X, Y, M, EIC, EOC, IC, SELECT >$

X: input events set;
Y: output events set;
M: set of all component models in DEVS;
$EIC \subseteq DN.IN * M.IN$: external input coupling relation;
$EOC \subseteq M.OUT * DN.OUT$: external output coupling relation;
$IC \subseteq M.OUT * M.IN$: internal coupling relation;
$SELECT$: $2^M - \emptyset \rightarrow M$: 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. Fig. 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.

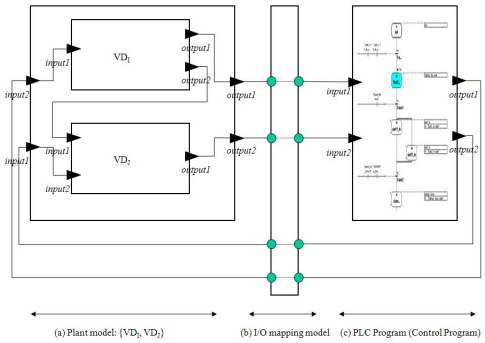

Figure 4: Interactions among three models of the PLC programming environment

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. The core part of a virtual device includes the inherent properties of the device, such as kinematics, geometric shape and the execution of device-level commands. For the modeling of the core part of a virtual device, the CSG (constructive solid geometry) modeling scheme was employed, as shown in Fig. 5. In the CSG modeling scheme, a user can interactively construct a solid model by combining various primitives, such as cylinders, spheres, boxes and cones. To imitate the kinematics of a real device, it is necessary to define the moving joints and the attributes of each joint. Our final goal is to provide a library containing all standard manufacturing devices, such as machining stations, AGVs, and robots. Therefore, users can easily instantiate virtual devices, without making efforts for modeling virtual devices.

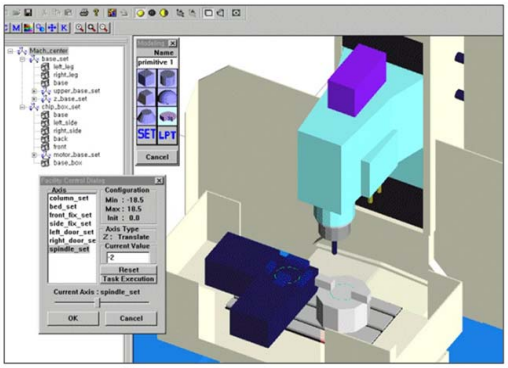

Figure 5: Core part modeling of a virtual device

The shell part, enclosing the core part, 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. Fig. 6-(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 Fig. 6-(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, Signla_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:

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

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

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

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

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

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

$$\lambda(DoT1) = T1Done$$

$$\lambda(DoT2) = T2Done$$

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

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

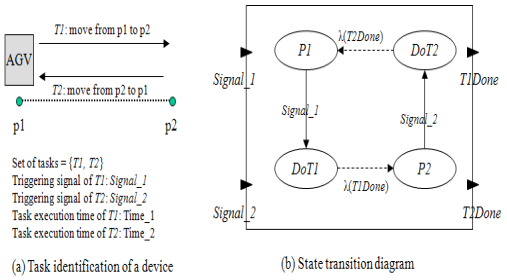

Figure 6: Shell modeling of a *virtual device*

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. Fig. 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 Fig. 4-(a), can be described as follows:

Plant Model:

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

The final objective of the plant model is the visual validation of a PLC program. To do so, it is necessary to define the communicating links between the plant model and the PLC program, which is described by the I/O mapping model. Fig. 7 shows the I/O mapping model for the AGV example shown in Fig. 6. Once the I/O mapping relations are established, the PLC program can control the plant model through the I/O mapping model.

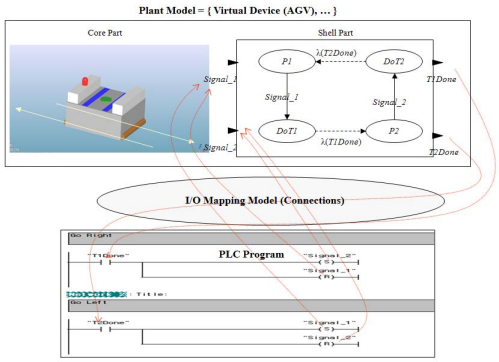

Figure 7: I/O mapping model between a plant model and a PLC program

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

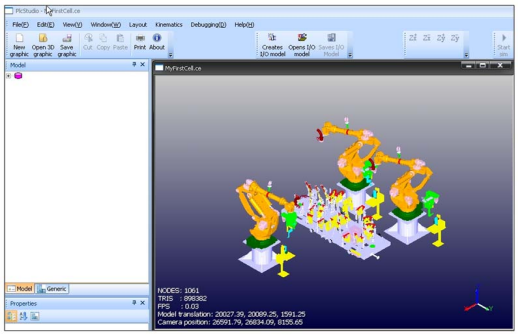

Figure 8: 3D graphic model of a welding Cell

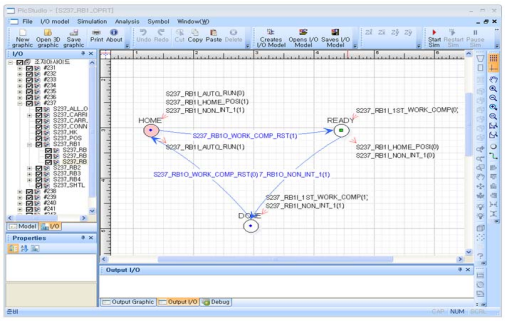

Figure 9: Plant model of a welding cell

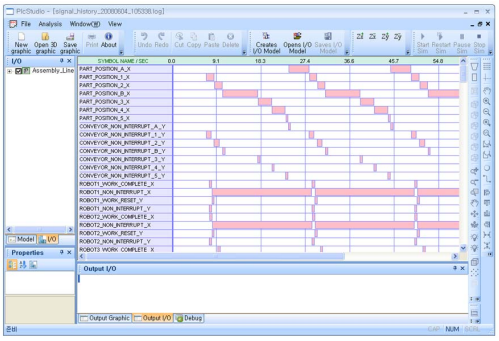

Figure 10: Gantt chart for checking simulation result

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 that we can intuitively define 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

C. P. Chuang, X. Lan, J. C. Chen, A systematic procedure for designing state combination circuits in PLCs, Journal of Industrial Technology, 1999;15(3):2-5.

S. Manesis, K. Akantziotis, Automated synthesis of ladder automation circuits based on state-diagrams, Advances in Engineering Software, 2005;36:225-233.

A. Rullan, Programmable logic controllers versus personal computers for process control, Computers and Industrial Engineering, 1997;33:421-424.

J. Jang, P. H. Koo, S. Y. Nof, Application of design and control tools in a multirobot cell, Computers and Industrial Engineering, 1997;32:89-100.

P. Klingstam, P. Gullander, Overview of simulation tools for computer-aided production engineering, Computers in Industry, 1999;38:173-186.

A. M. A. Al-Ahmari, K. Ridgway, An integrated modeling method to support manufacturing system analysis and design, Computers in Industry, 1999;38:225-238.

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

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

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

B.K. Choi, B.H. Kim, 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, 2000, 425-436

M. Onosato, K. Iwata, Development of a virtual manufacturing system by integrating product models and factory models, CIRP, 1993;42(1):475-478.

K. Iwata, M. Onosato, K. Teramoto, S. Osaki, A modeling and simulation architecture for virtual manufacturing systems, CIRP, 1995;44(1):399-402.

L. Ye, F. Lin, Virtual system simulation – A step beyond the conventional simulation, *22nd Int. Conf. on Computer and Industrial Engineering*, 1997/12/20, 304-306

Anglani A, Grieco A, Pacella M, Tolio T. Object-oriented modeling and simulation of flexible manufacturing system: a rule-based procedure, Simulation Modeling Practice and Theory, 2002;10:209-234.

G. Fray, Automatic implementation of Petri net based control algorithms on PLCs, Proceedings of the American control conference ACC 2000, Chicago; 2000, 2819-23.

J. S. Lee, P. L. Hsu, A PLC-based design for the sequence controller in discrete event systems, Proceedings of the 2000 IEEE International conference on Control Applications, Anchorage; 2000, 929-34.

M. A. Jafari, T. O. Boucher, A rule-based system for generating a ladder logic control program from a high level system model, Journal of Intelligent Manufacturing Systems, 1994 ;5 :103-120.

D. Alexandre, B. Gerd, G. L. Kim, Y. Wang, A tool architecture for the next generation of UPPAAL, http://www.uppaal.com/ ; 2003

G. L. Kim, P. Paul, Y. Wang, UPPAAL in a nutshell, International Journal on Software Tools for Technology Transfer, 1997;1(1+2):134-152.

AUTHOR BIOGRAPHIES

S.C. PARK is an assistant professor in the Department of Industrial & Information Systems Engineering at Ajou University. Before joining Ajou, he worked for DaimlerChrysler Corp. and CubickTek Co., developing commercial and in-house CAD/CAM/CAPP/simulation software systems. He received his BS, MS, and PhD degrees from KAIST in 1994, 1996, and 2000, respectively, all in industrial engineering. His research interests include geometric algorithms in CAD/CAM, process planning, engineering knowledge management, and discrete event system simulation. He can be reached via email at <mailto:scpark@ajou.ac.kr>.

Gi-NAM WANG is a department head and professor in the Department of Industrial & Information Systems Engineering at AJOU University, South Korea. He has completed his PhD in 1992 from Texas A&M University, in Industrial Engineering. He has worked as visiting professor at University of Texas at Austin during 2000-2001. His area of research is related to Intelligent Information & manufacturing system, system integration & automation, e-Business solutions and image processing. He can be reached via email at <mailto:gnwang@ajou.ac.kr>.

CHANG MOK PARK is a research professor in the Department of Industrial & Information Systems Engineering at AJOU University. He has completed his PhD in 2002 from AJOU University, in Industrial Engineering. His research interest is related to Intelligent Information & manufacturing system, and image processing. He can be reached via email at <mailto:cmpark@ajou.ac.kr>.

JONGEUN KWAK is PHD student in Industrial and Information Systems Engineering at Ajou University. He did his BS and MS in Computer Science Department in year 1994 and 1996 respectively, from POSTECH, Korea. Before joining PHD, he worked in LG Cooperative Institute of Technology as a senior researcher for 5 years, he also worked in Unisoft Ltd, and at Freelancer until 2007. He can be reached vial email at <mailto:jkwak@ajou.ac.kr>.

SUNGJOO YEO is PHD student in the Department of Industrial and Information Systems Engineering at Ajou University. He completed his BS, MS in year 1999 and 2001 respectively in Ajou University, IISE Department. Before joining PHD, he worked in FUJITSU LIMITED as a senior IT consultant for 7 years. His research area includes simulation and data mining. He can be contacted via <mailto:oriheap@ajou.ac.kr>.