# A Joint Controller-Simulator Programming by Demonstration Method for PLC Devices

Phissanu Changsakol, Nopporn Chotikakamthorn, Olarn Wongwirat
King Mongkut's Institute of Technology Ladkrabang
Faculty of Information Technology and Research Center for Communication and Information Technology
Bangkok, Thailand
Phissanu@hotmail.com

*Abstract*— **In this paper, a problem of visual programming of PLC (programmable logic controller) devices is described. A programming by demonstration method is developed so that a problem-domain expert can program such devices without computer programming skill. The method is novel in the sense that, unlike other programming-by-demonstration methods, it requires little a priori knowledge of the controlled system model, nor it needs to interact with a real system. A simulation model, which captures the laws of nature governing the system behavior, is jointly programmed with a control device (PLC). A PLC programming tool, developed based on the proposed joint controller-simulator programming by demonstration paradigm, can generate a high-level structure text program code, from the demonstrated actions as performed by a user.**

*Keywords*—**Programming by Demonstration, Programmable Logic Controller, Programming Paradigm, End-User Programming**

## I. INTRODUCTION

Programmable Logic Controller (PLC) is a digitally operating electronic apparatus which uses a programmable memory for the internal storage of instructions. It includes functions such as logic sequencing, timing, counting, and arithmetic operations, to control various types of machines or processes. PLCs programs are generally written by a specialized programming language as specified by the IEC 1131-3 standard. It is thus not possible for an end user to program the devices. Programming by demonstration (PBD) concerns with providing a non-programmer tools and techniques to control, automate, and customize the software applications or intelligent physical devices. PBD technique is simpler than procedural programming because there is no need to learn a language syntax and rules. This concept has been applied for various purposes such as those listed below.

- Automation of repetitive tasks using software functions. For example, a macro recorder learns patterns of user's activity and can then execute those patterns autonomously. [1, 5]
- User interface customization and personalization. Here, PBD is applied to change interface appearance to optimize access to frequently used functions. [6]
- PBD is also applied to robotic manipulation. In [2, 4], a PBD technique is proposed. so that a robot arm movement can be programmed through demonstration.

One of the most important limitations of those existing programming-by-demonstration methods, when applied to a problem of PLC programming, is the need for an existence of a real system to accompany a demonstration. Alternatively, without a real system, a complete simulation model of the system must be available so that a user can perform a demonstration on a system simulator instead. In this paper, a programming-by-demonstration method is developed to avoid such limitation. Without the need for a priori knowledge on the system simulation model, the method shares a similar generality property of standard programming languages.

The paper is divided into two parts. First, PLC programming paradigms are described. Next, the proposed joint controller-simulator programming by demonstration method is detailed in Sections 3 and 4. Use of the PLC programming tool, developed based on the proposed programming paradigm, is described using a simple water tank system as a case study. Finally, concluding remarks are given.

## II. PROGRAMMING PARADIGMS FOR PLC DEVICES

Programming paradigm is a paradigmatic style of programming. It provides and determines the view that the programmer has of the execution of the program. For examples, in object-oriented programming, programmers can think of a program as a collection of interacting objects, while in functional programming it can be thought of as a sequence of stateless function evaluations. The common PLC programming paradigms are described below.

*1) Conventional programming paradigm:* This is a programming paradigm based upon the concept of procedures, also known as routines, subroutines, methods, or functions, that contain a series of computational steps. Examples of PLC languages under this conventional programming paradigm category are mnemonic and ST (Structure text) language.

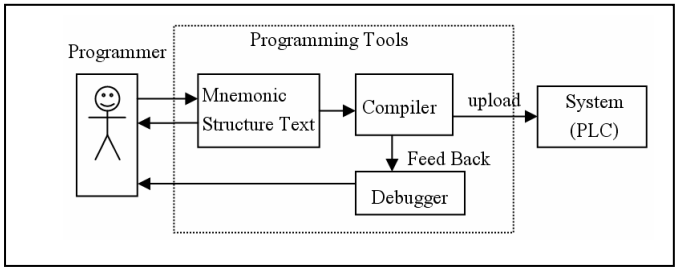


Figure 1. A block diagram of PLC programming under the conventional programming paradigm

*2) Visual programming paradigm:* This is a programming language that uses a visual representation such as graphics, drawing, animation or icons, partially or completely. It views program construction as a visual task similar to the solving of jigsaw puzzles. Example of the language that uses this programming paradigm includes the ladder diagram, the function block, and the sequential function chart.

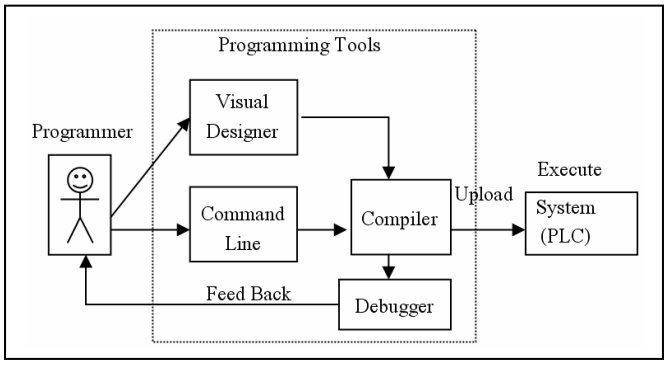


Figure 2. A block diagram of PLC programming under the visual programming paradigm

Programming by Demonstrations (PBD) is the most recent approach to programming and is largely used at present for end-user programming. The essential notion is that the user demonstrates a program to the system by working through one or more examples. The system then generalizes the users' sequence of actions to produce a general program. One of the best known such systems is Cypher's EAGER system which can automatically detect repetitions in user's actions and build a program to perform the use task. [11]

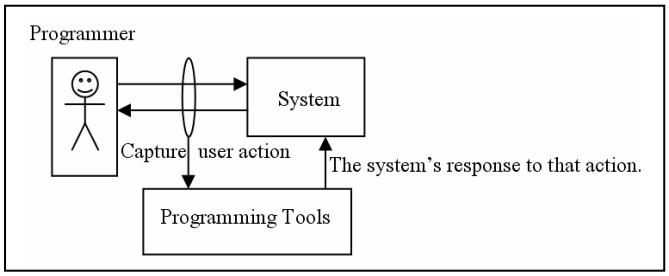


Figure 3. A conceptual view of the programming by demonstration paradigm

The programming by demonstration paradigm is generally applicable only when the full functional system (to be programmed) is available. Therefore, applicability of PBD to program a new system that has not yet existed is questionable. Similarly, applying PBD concept to PLC programming requires the existence of a controlled system to be interacted with during the demonstration.

In this paper, the above problem is solved by a joint controller-simulator programming by demonstration method. This technique combines visual programming and PBD for jointly programming of controller devices and a simulation model of the controlled devices. A joint controller-simulator programming by demonstration method is described in the next section.

## III. A JOINT CONTROLLER-SIMULATOR PROGRAMMING BY DEMONSTRATION METHOD FOR PLC DEVICES

A joint controller-simulator programming by demonstration method is composed of four parts as follows (see Figure 4):

- Model builder: This module is responsible for building relationships among controller devices and controlled system.
- System simulator: This module provides an animated display of the controlled system dynamics in response to a controller action.
- User action recorder: This module records a user action for the purpose of both programming the controlled system simulation model and the controller.
- Compiler: This module is used to transform a recorded user action into an actual PLC programming code. The code can be in a high level language such as a structure text, or a low-level one such as a mnemonic code.

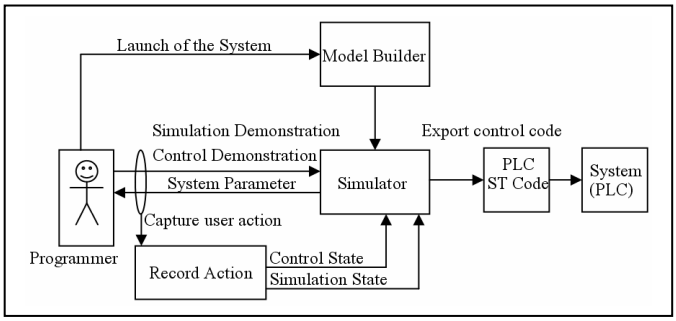


Figure 4. Main modules of the joint controller-simulator programming method

Conceptually, the system as shown in Figure 4 operates in one of five modes (states) as explained below (see Figure 5).

*1) System initialization mode:* In this mode, a controlled system is created by a user through visual interface. Each controlled device and the corresponding properties can be defined. In addition, through visual programming, general

relationship among controlled devices, and between controlled devices and the controller devices can be declared (visually).

*2) Simulator programming mode:* Under this mode, a system simulation model, which captures the relationship among controlling and controlled devices, can be programmed through user demonstration. A user programs the model by demonstrating how controlled device's properties change in response to a controller action, as well as to changes in other controlled devices' properties.

*3) Controller programming mode:* In this mode, a user is allowed to define a controller action in response to a particular system state. As for the case of the simulator programming mode, a controller is programmed through user demonstration of actions.

*4) Simulation mode:* In this mode, a programming tool demonstrates to a user how a system state changes in response to a controller action and vice versa. The simulation is provided based on a system simulation model, constructed from previous user actions under Mode 2. Therefore, the model may be incomplete and the simulation may be inaccurate. System model and simulation accuracy can be corrected, however, through iteratively re-programming.

*5) Compiler mode:* When in this mode, the tool transforms user actions into an actual PLC programming code. For example, the tool developed in this study chooses ST (Structure Text) as specified by the IEC1131-3 standard as a targeted PLC programming language.

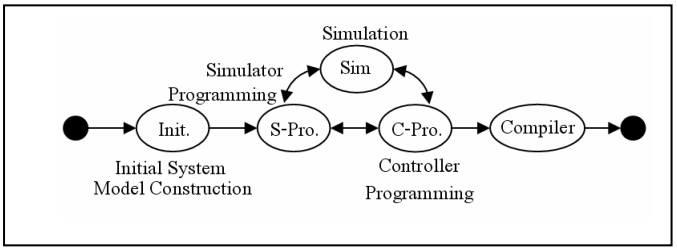


Figure 5. Programming modes of the proposed method

## IV. PBD-BASED PLC PROGRAMMING TOOL

In this section, design of the tool developed based on the proposed method as described in Section 3 is described below.

*1) Model builder:* This module is active when the tool is in the system initialization mode. It is used for system model construction. Using a visual programming paradigm, a controlled and/or controller device is defined by the selection of icons from the tool palette. Each device's properties are defined through the object inspector (see Figure6).

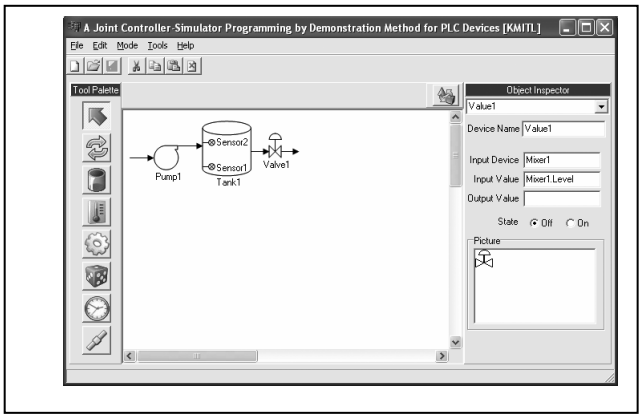


Figure 6. Screen shot of the tool's model builder

TABLE I. STANDARD OBJECT CLASSES SUPPORTED BY THE TOOL

| ***Device*** | ***Description*** |
|---|---|
| 1. Activator | A generic controller's output device |
| 2.Sensor | A generic controller's input device |
| 3.Container | A generic controlled device |
| 4.Counter | A specialized counter device |
| 5.Timer | A specialized timer device |
| 6.Generic | A general purpose device class for extension |
| 7. PipeLine | A Connector |

TABLE I summarizes standard device classes as supported by the developed tool.

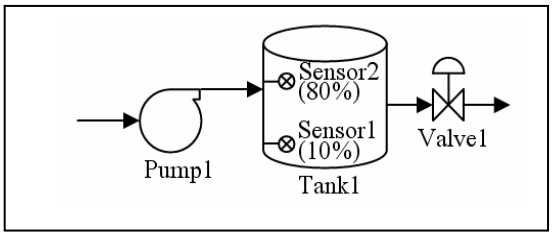


Figure 7. A simple water tank controlling system

To illustrate how the tool is used for PLC programming, a simple water tank controlling system shown in Figure 7 is used as a case study. To create a simulator model, relationship between controlling device and controlled device as well as the relation between controlled device and controlled device need to be defined. Through the language of visual programming, two types of relationship can be defined.

*a) Association relationship:* Once defined, allows the associated objects to exchange message (parameters, commands, etc.) with each other. For examples, in our case study, when Valve1 is connected with Tank1 through the connector device, opening or closing of Valve1 can have an

effect on the properties (such as a water level) of Tank1. The direction of the connector's arrow head is used to identify the direction of message flow.

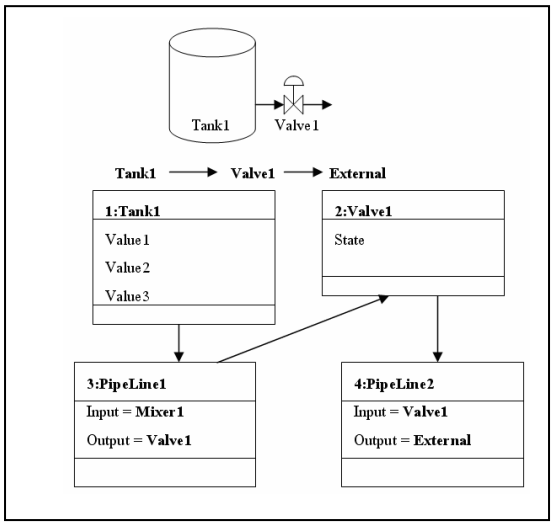


Figure 8. Defining the association relationship

*b) Dependency relationship: I*s applied to define a part-of relationship between two or more controlled devices, or between a controlled device and one or more controller devices. For example, in our case study, to install a controller sensor to measure a tank's water level, the sensor is placed inside the container Tank1 to establish the required relationship. In Figure 9, two such sensors, L1 and L2, are defined with this type of relationship with Tank1, so that two (low and high) water levels can be sensed.

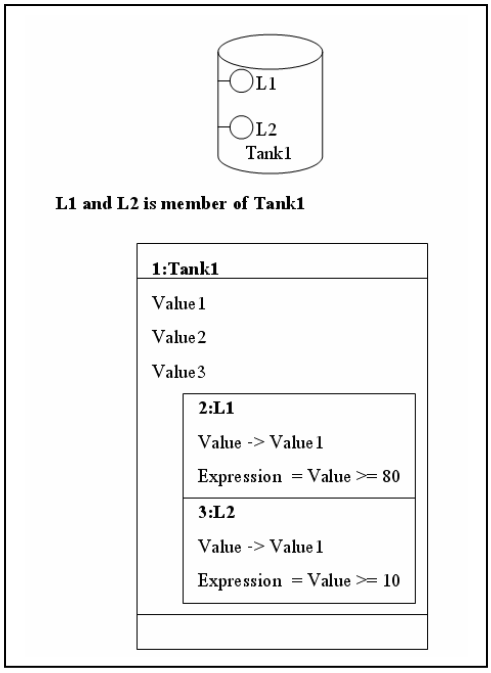


Figure 9. Defining the dependency relationship

*2) User action recorder:* This module is responsible to recording a user action. It operates in one of the two modes, the simulator and controller programming modes. The two modes, along with the simulation mode, can be inter-changed. Generally, the simulator programming mode is activated when a user changes the state of a controlling device of which the corresponding system response is not known yet. Once a system state change in response to the controller action is defined during the simulation programming mode, the tool should be switched to the simulation mode. From then, when a particular system state occurs, a user can switch to the controller programming mode so that an appropriate controller action can be defined.

To illustrate how to program a system simulation mode, consider our case when Valve1 or Pump1 is turned on/off (Figure 10).

*a)* In response to the controller state change when the status of Pump1 is turned to "ON", a user switches to the mode. Based on the association relationship between Pump1 and Tank1 as established through visual programming during system model initialization, the tool prompts for a change to any of Tank1's properties. A user can then program the simulation model by raising the tank's water level. The tool records the action, with a dialogue for the user to define the rate of water level increase.

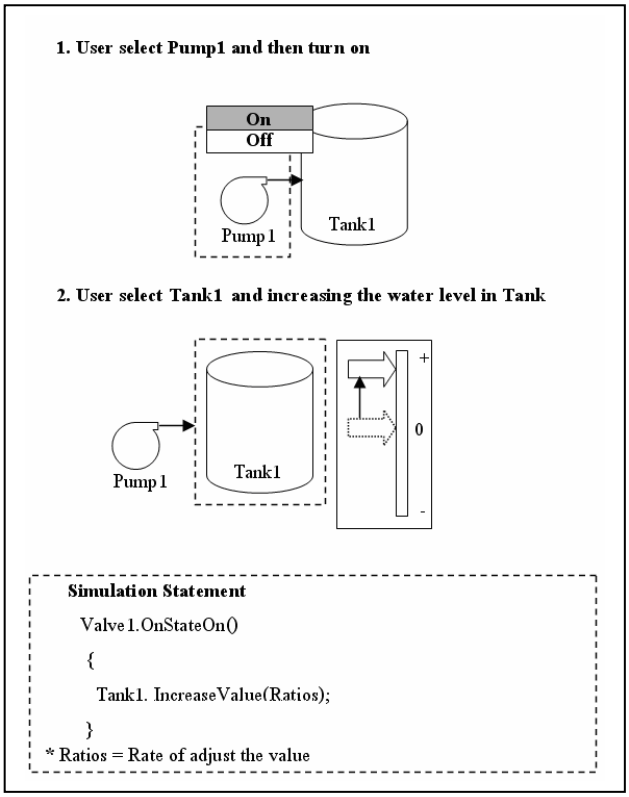


Figure 10. Using the tool to program the system model when Pump1 is turned on

*b)* Consider then the case where Valve1 is turned on. When switched to the simulator programming mode, Tank1 is highlighted as a result of the established relationship between Valve1 and Tank1.To drain the water level in the tank, a user select the tank's property that is affected by this controller action and then lowering the water level. The tool then records the action (see Figure 11).

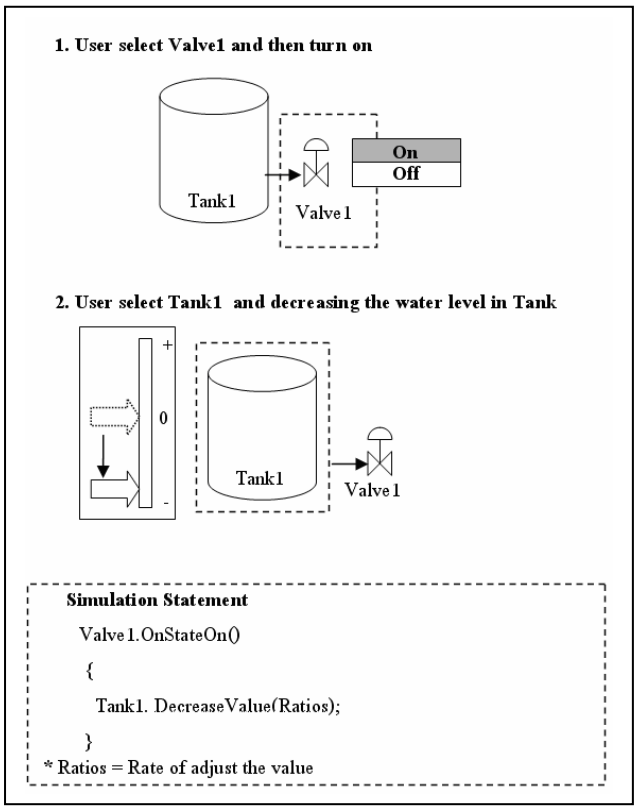


Figure 11. Using the tool to program the system model when Valve1 is turned on

*3) Controller Porgramming:* To illustrate how to program a controller through demonstration, consider the following scenarios.

*a)* While in a simulation mode, a user notices that a water level is increasing. Once the level arrives at the point specified by Sensor 1, the simulator then prompts a user for a controller action by switching to a controller programming mode. The user informs the tool to continue the simulation. When the level arrives at the point specified by Sensor 2, the simulator prompts for the action again. This time, however, the user defines the action by visually turning Pump1 off. The tool then records that action (and if no other action, the tool switches to the simulator programming mode).

*b)* While in a simulation mode, the water level is decreasing. When it passes the level below that specified by Sensor1, a user can proactively turns the tool into a controller programming mode himself. He then can program the controller by visually turning Pump1 on (see Figure 12).

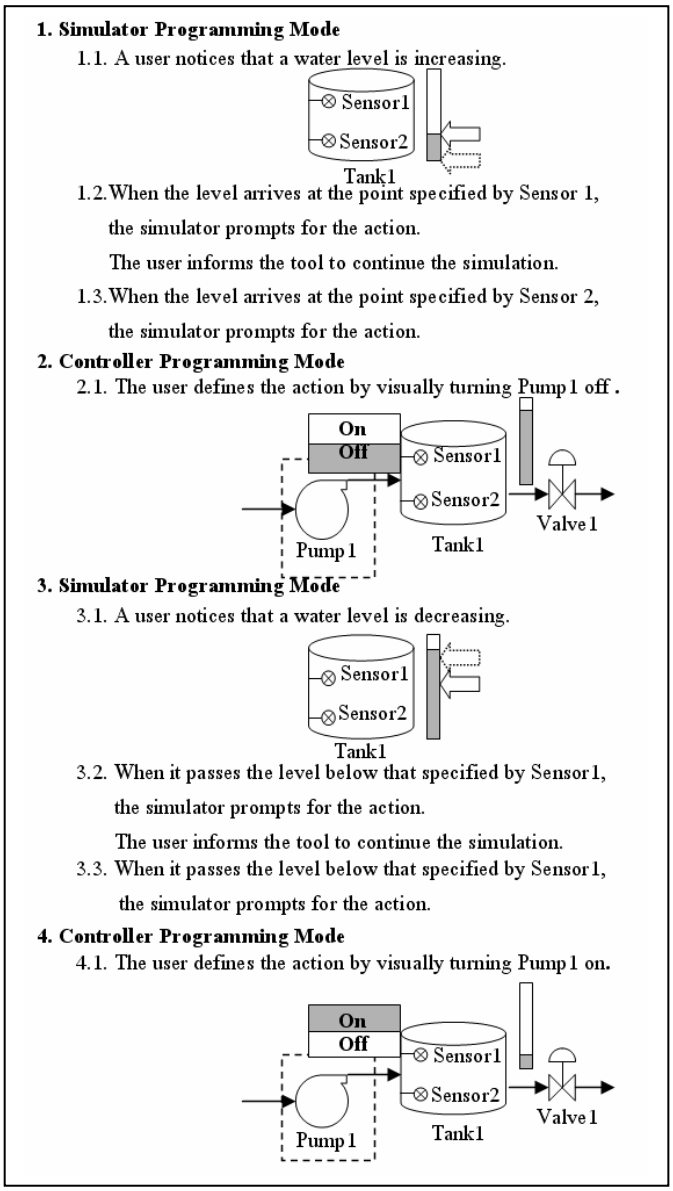


Figure 12. Programming the controller during the controller programming mode

Figure 13 describes the sequence of user actions to program both the simulator and the controller. From the figure, note the iterative nature of the joint-programming process.

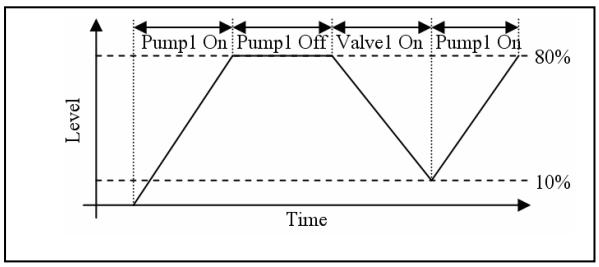


Figure 13. A sequence of user actions during the joint programming-by-demonstration process

From the changes in controller devices' states as described, those involved devices' states can be summarized as shown in TABLE II.

TABLE II. CONTROLLER AND CONTROLLED DEVICES' STATE CHANGES

| *No* | *Sensor1* | *Sensor2* | *Pump1* |
|---|---|---|---|
| 1 | 0 | 0 | 1 |
| 2 | 1 | 0 | X |
| 3 | 1 | 1 | 0 |
| 4 | 1 | 0 | X |
| 5 | 0 | 0 | 1 |

0 = State Off, 1=State On, X=User Ignore.

From the above state table, a truth table describing all possible logic combinations of the controller devices' states can be derived as shown in TABLE III. In this case, user action scenarios cover all possible combinations. Therefore, the truth table is simply the state table with redundant row being removed. When user action scenarios do not cover all possible combinations, the tool must ask the user for more information by displaying the missing scenarios. The topic of generating a missing scenario is discussed elsewhere, however.

TABLE III. A TRUTH TABLE AS GENERATED FROM THE STATE TABLE

| *Sensor1* | *Sensor2* | *Pump1* |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | - |
| 1 | 0 | X |
| 1 | 1 | 1 |

0 = State Off, 1=State On, X=User Ignore, - = Unknown.

From the truth table (see TABLE III), an actual controller program can be generated when the tool is switched to the compiler mode. For the case of our simple water tank system, the generated code for each scenario as explained so far is summarized in TABLE IV.

TABLE IV. CODES GENERATED CORRESPONDING TO VISITED SCENARIOS AND USER DEMONSTRATIONS

| *Scenario* | *ST generated code* |
|---|---|
| Sensor Sensor1;<br>Sensor Sensor2; | INPUT_VAR<br>Sensor1:BOOL;<br>Sensor2:BOOL;<br>END_VAR |
| Activator Pump1; | OUTPUT_VAR<br>Pump1:BOOL;<br>END_VAR |
| IF(Sensor1.State = Off &<br>Sensor2.State = Off)<br>Pump1.State = On; | IF Sensor1 = FALSE AND Sensor2 = FALSE THEN Pump1 :=TRUE;<br>END_IF; |
| IF(Sensor1.State = On &<br>Sensor2.State = On)<br>Pump1.State = Off; | IF Sensor1 = TRUE AND Sensor2 = TRUE THEN Pump1 :=FALSE;<br>END_IF; |

## V. CONCLUDING REMARKS

In this paper, the problem of PLC programming by the programming by demonstration paradigm has been considered. A new joint controller-simulator programming-by-demonstration method has been proposed. Details of the design and implementation of the PLC programming tool based on the proposed method has been given. Use of the tool for PLC programming has been demonstrated by using a simple case of a water tank system. Based on user demonstration, the tool can generate a PLC programming code in a form of a structure text as defined by IEC 1131-3 standard.

## REFERENCES

[1] Gordon W. Paynter and Ian H. Witten, "Applying machine learning to programming by demonstration", University of California, Riverside and University of Waikato New Zealand, 2004.

[2] Haiying She and Axel Graeser, "Closed Loop Control and Automatic Set Point Generation in Programming by Demonstration for Service Robotic Tasks", University of Bremen, Germany, 2004.

[3] Richard M. Voyles, "Toward Gesture-Based Programming: Agent-Based Haptic Skill Acquisition and Interpretation", University of Minnesota, 1997.

[4] S. Munch, J. Kreuziger, M. Kaiser, R. Dillmann , "Robot Programming by Demonstration (RPD) Using Machine Learning and User Interaction Methods for the Development of Easy and Comfortable Robot Programming System", University of Karlsruhe, Germany, 1994.

[5] Tessa Lau, "Programming by Demonstration: a Machine Learning Approach", University of Washington, 2001.

[6] Lawrence D.Bergman, Tessa A. Lau, Vittrio Castelli, Daniel Oblinger, "Programming-by-Demonstration for Behavior-based User Interface Customization", IBM T.J. Watson research Center, USA, 2004.

[7] Henry Lieberman, "Your Wish is My Command: Giving Users the Power to Instruct their Software", Massachusetts Institute of Technology, 2000.

[8] Richard G. McDaniel, "Creating Complete User Interfaces by Demonstration", Carnegie Mellon University, 1993.

[9] Ben Shneiderman, "Foreword to Your Wish is My Command", University of Maryland, 2000.

[10] Karl-Heinz john and Michael Tiegelkamp, "ICE 61131-3 Programming Industrial Automation Systems", Springer, january 2001.

[11] Cypher, Allen, Introduction, In Cypher, Allen, ed., "Watch What I Do: Programming by Demonstration", Cambridge: The MIT Press, 1993.