EAI Endorsed Transactions
on Digital Transformation of Industrial Processes Research Article EAI.EU

# Cloud-Based Data-Driven Behavior Model Recovery for Distributed Automation Systems

Xian Wu[1], Chuanyang Yu[2,3], Likuan Zhang[2,3], Hui Zhang[2,3] and Wenbin Dai[1,3*]

[1]Shanghai Jiao Tong University, Chima, 800 Dongchuan Road, Minhang District, Shanghai, China
[2]Beijing Urban Construction Intelligent Control Co., Ltd, China, Building 4, Courtyard 2, Liulijing Road, Yongdingmenwai Street, Dongcheng District, Beijing, China.
[3]Shanghai Jiao Tong University School of Electrical, Information and Electronic Engineering-Beijing Urban Construction Intelligent Control Joint Research Lab on Universal Automation.

## Abstract

INTRODUCTION: Industrial cyber-physical systems provide a bridge between legacy controllers and new edge devices that are usually equipped with massive computing power and storage capacity. The migration from legacy control systems to industrial cyber-physical systems is facing challenges as control code and design documents of legacy systems may not be available.
OBJECTIVES: This paper proposes a data-driven behavior model recovery method for the black-box distributed manufacturing system based on cloud computing.
METHODS: This method adopts the IEC 61499 function blocks as meta-models to describe system behaviors. The proposed framework includes three parts: data mining, logic restoration, and application construction. Raw collected data are processed and encapsulated into function block sets, then execution control charts, and finally function block types.
RESULTS: This model recovery method is validated with a process control system of the food and beverage industry.
CONCLUSION: A deployable function block network is generated by instantiating and connecting these function blocks.

**Keywords:** Model-Driven Engineering, IEC 61499 Function Blocks, Model Recovery, Finite-State Machine, Data-Driven Model Generation

Received on 02 February 2025, accepted on 16 March 2025, published on 19 March 2025



doi: 10.4108/_____

## 1. Introduction

Industrial automation systems are shifting from massive production to massive customization by introducing the industrial cyber-physical system (ICPS) [1]. The ICPS aims to improve flexibility and interoperability by enabling information exchange between various field devices. On the other hand, the majority of legacy automation systems are still running as black boxes and cannot be easily migrated. One feasible solution is to recover the control software model from operating data based on the model-driven engineering (MDE) technology [2]. The MDE is commonly used in automatic code generation to improve the efficiency and quality of the software development process [3].

In the MDE, software behavior models (platform-independent) are transformed into executable code (platform-dependent) by using model transformation methods. However, it is extremely difficult to update the behavior model using rule-based algorithms to capture changes made in the code. One reason is that the code pattern may be changed completely and no pre-defined rule can be matched. As a result, even with a simple code change request, complete regeneration of the entire code is compulsory which is extremely inefficient.

*Corresponding author. Email: w.dai@sjtu.edu.cn

Another approach is to recover the software behavior model by using the data-driven approach. These approaches are based on pre-defined models or meta-model templates generated from data to refactor executable modules [4][5]. The major challenge is to capture control logic and data structure from system inputs and outputs. The internal logic in the controller remains as a black box. With an unknown internal structure, it is challenging to compose a behavior model that is 100% equivalent to the original model.

Cloud computing bridges distributed system behavior models with real-time data collected from controllers on the shop floor. With a large process automation system like nuclear plants, the number of I/Os and variables could generate 10MB per I/O scan (in tens of milliseconds). The data-driven behavior model recovery method utilizes cloud computing power to classify massive historical data into various software components and regenerate internal logic for each component. The overall system behavior is characterized as an ordered execution sequence of process operations. These components are composed of a complete application with support from domain-specific models.

This paper's main contribution is to propose a framework to recover the system behavior model from historical operation data for migrating legacy automation systems without source code available. The behavior model is reconstructed as an IEC 61499 function block (FB) network with state-machine encapsulated in FBs to provide human-understandable system behaviors with hierarchical levels of components for abstraction. The IEC 61499 standard offers a component-based open architecture for distributed automation systems in which a distributed application can be composed of event-driven FBs. The IEC 61499 standard [6] has been proven a perfect fit for rapid prototyping distributed automation systems. The function block network can be directly deployed and executed on a group of connected devices to improve the efficiency of migration processes [7]. Several data processing steps are proposed to regenerate IEC 61499-based system behavior model. As a result, the control logic in the black box can be migrated to any new edge computing device with minimum effort.

The rest of this paper is structured as follows. Section II provides a review of related works for MDE and behavior model recovery. In Section III, details of the behavior model recovery method are discussed. Section IV demonstrates the proposed method with a dairy production process. Finally, Section V concludes the paper and provides future research directions.

## 2. Related Works

The key to accelerating cloud-edge collaborated ICPS is the migration of legacy control systems. Currently, industrial automation control systems are programmed according to the IEC 61131-3 standard [8]. The IEC 61131-3 programmable logic controllers (PLCs) are widely used but are limited to the scope of a single device without a clear system-level view and interoperability between multiple devices. There is no existing work on data-driven recovering control logic for IEC 61499-based control applications. However, some existing data mining and model recovery algorithms were used during the I/O partitioning process.

A framework was proposed for automatic reverse engineering binary codes from industrial control systems by Keliris et al [9]. A structured methodology was used to extract information including header, subroutine, symbol tables, and data sections from PLCs. The original program structure was recovered by detecting subroutines from binary code. Also, the pattern of known function blocks was matched. Keller et. al. [10] proposed a pattern-based reverse engineering method with abstract UML metamodels and source code. Three different methods including template, factory, and bridge were compared. Manual efforts were still required for pattern recognition. A framework that recovers execution semantics from embedded IoT controller software binaries was proposed by Sun et. al [11]. Control Flow Graph and Symbolic Controller abstraction methods were used to recover semantic models from code binaries. The approach still required domain-specific knowledge.

There are lots of existing works on the migration of legacy control systems into IEC 61499 system configurations based on model-driven engineering (MDE) approaches. The key concept of MDE is to employ high-level, domain-specific models in the implementation, integration, maintenance, and testing during the full life cycle of software systems. The IEC 61499 standard is a typical MDE practice that uses event-trigger function blocks as the basic programming organization unit. The function block defined in this standard is a software entity that encapsulates algorithms with a generic interface that consists of event and data variables. These FBs are suitable for the system-level design for ICPS [12]. Currently, most model-driven design patterns of the IEC 61499 standard focus on the organization of the function blocks at the application level [13][14].

Many researchers propose migration and transformation rules by encapsulating existing Structured Text (ST) based IEC 61131–3 functions in IEC 61499 FBs [15] and integrating IEC 61131-3 execution into IEC 61499 FBs [16]. These methods mainly focus on module conversion with embedded code that the source code of the legacy system must be provided.

Chivilikhin et al. [17] proposed a framework that generates state machines that can be incorporated into IEC 61499 FBs from collected PLC data when the original program is unavailable. This framework takes the trace of raw PLC data and produces a set of candidate state machines using Boolean satisfiability solvers. It synthesizes a global controller by modular decomposition. This method is exemplified in a grasping system that contains only Boolean input/output variables from a single controller with no hierarchy.

Ladiges et al. [18] proposed learning behavior models of discrete event production systems based on I/O signals. A machine state presented by Petri Nets was generated from recorded event traces. All I/O signals were expressed in Boolean variables and timing information was considered. Further, these state machines were automatically organized into material flow also presented in Petri Nets [19].

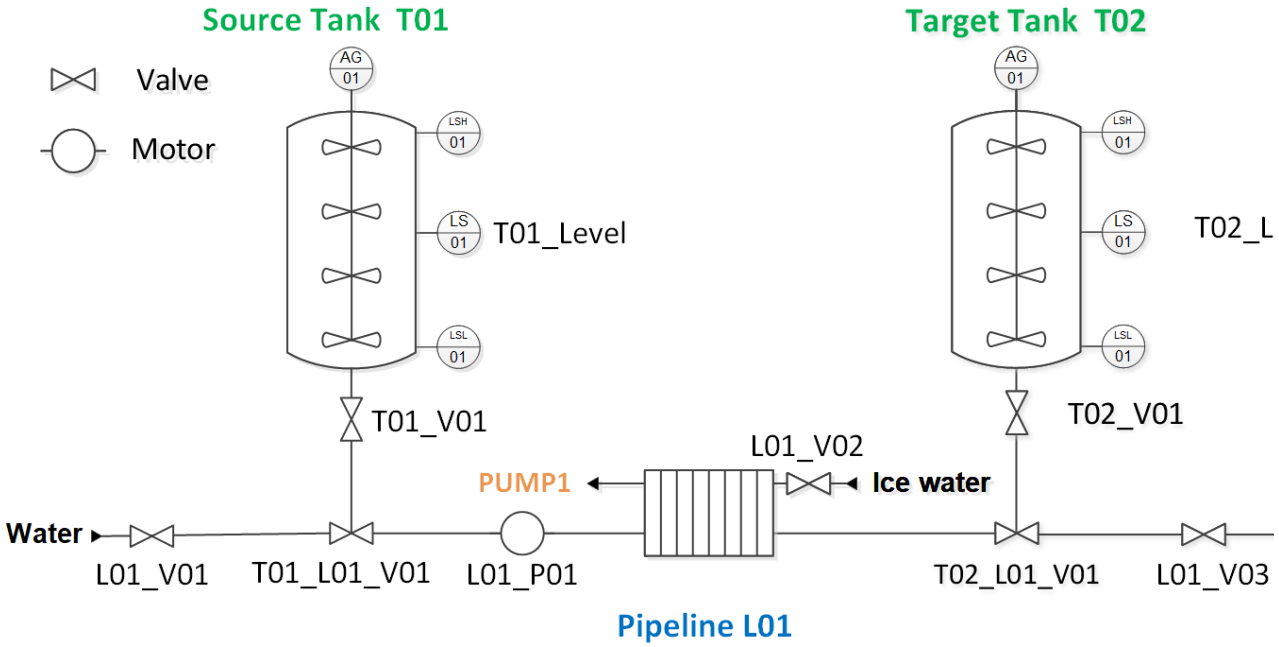


**Figure 2** Process & Instrument Diagram of the TLT model

Similar to the previous work, Lesage et al. [20] generated interpreted Petri net models of the discrete event manufacturing systems from observed input/output (I/O) signal vectors. Direct relations between I/O events and the internal state evolutions are identified. Besides, an algorithm inspired by clustering methods is proposed to partition a complex distributed system [21]. This research is highly relevant to the restoration of legacy system behavior models, although the generated Petri net was not directly executable code and I/O data were limited to Boolean variables.

This paper proposes a methodology suited for legacy automation systems with linear process flow with several independent workstations. These workstations can be operated in a certain order to process a dedicated task within a fixed period. During this period, products are transported between workstations to complete different processes at each workstation. The model analysis of non-Boolean variables and the relevance of time-shift sequences are covered. Besides, specific data changes are defined as events. Compared to existing works, this approach uses IEC 61499 FB network as the recovered model to ensure all behaviors are completely modularized. This encapsulation provides better reusability and software components recovered from the data can be deployed directly to any device without any code modification.

## 3. Behavior Model Recovery

As shown in Fig. 1, the behavior model recovery process contains three modules: data mining, logic restoration, and application reconstruction. Raw data are collected by directly reading values from legacy Programmable Logic Controllers (PLC) for every I/O scan cycle. These data cached on edge gateways are transferred back to the cloud and clustered as independent variable sets. Then these variable sets are reconstructed as finite-state machines (Execution Control Charts in IEC 61499) encapsulated by function blocks. Challenges include data redundancy, state-space explosion, and uncertain system structure.

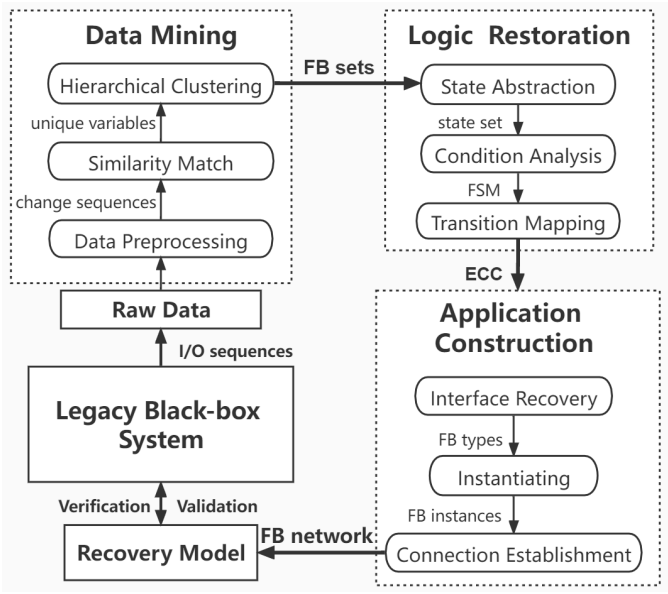


**Figure 1** The Cloud-based behavior model recovery framework for distributed control system

According to ANSI/ISA S88 standard, a procedure system is a typical procedural and divisible system that directs separate equipment-oriented actions to take place in an

ordered sequence [22]. To balance multi-usability and domain-specification, this section applies a Tank-Link-Tank (TLT) model shown in Fig. 2 as an illustrative example. The TLT model is a design paradigm commonly used in the pipeline process which is suitable for most process control systems, especially for the Food and Beverage (F&B) industry.

This system is controlled by Schneider Electric Modicon M251 PLCs with a program the cycle time is set to 100ms. No data packets got lost during the transmission and only changes in the I/O sequences are recorded. Table I lists all collected data to the cloud of three equipment modules and functional system inputs.

TABLE I
COLLECTED DATA OF THE TLT SYSTEM

| *Eqpt* | *Variable* | *I/O* | *Description* |
|---|---|---|---|
| T01 | $T01\_Level$ | - | Liquid level sensing variable (Liter) |
| | $T01\_V01$ | O | Valve control |
| T02 | $T02\_Level$ | - | Liquid level sensing variable (Liter) |
| | $T02\_V01$ | O | Valve control |
| L01 | $T01\_L01\_V01$ | O | Valve control connecting T01 and L01 |
| | $T02\_L01\_V01$ | O | Valve control connecting T02 and L01 |
| | $L01\_V01$ | O | Valve control of washing water inlet |
| | $L01\_V02$ | O | Valve control of ice water inlet |
| | $L01\_V03$ | O | Valve control of pipeline outlet |
| | $L01\_P01$ | O | Motor control of the pump |
| | $L01\_Flow$ | - | Flow sensor variable (L/s) |
| System | $START$ | I | User input start signal |
| | $FINISH$ | I | User input finish signal |
| | $STOP$ | I | User input stop signal |
| | $srcTankID$ | I | User input source tank ID |
| | $tgtTankID$ | I | User input target tank ID |
| | $LineID$ | I | User input pipeline tank ID |
| | $Amount$ | I | Required material amount |
| | $Timer$ | I | System clock |

## 3.1 I/O Data Partitioning

The IEC 61131-3 defines elementary data types such as *BOOL, INT, REAL*, etc [8]. Non-boolean variables are classified and decomposed into Boolean signals representing device or process states.

For the TLT model, control signals of valves and motors are defined as on-off *BOOL* variables. Sensing variables are observed *REAL* type state signals, and system inputs are periodic constants except for the internal clock *Timer*. Non-boolean state variables (e.g., sensing variables, feedback variables) are abstracted as Boolean signals representing states. For example, liquid level sensing variables of Tank T01 are decomposed as *T01_level_up* and *T01_level_down*, which indicate the rising and falling state of the liquid level. Non-boolean condition variables (e.g., control variables, timing variables) are decomposed into multiple event sequences. For example, the *Timer* in the TLT model is decomposed into three event sequences (*Timer_20s*, *Timer_15s*, and *Timer_12s*) since three timing cycles are built into the system clock. The simplified I/O sequences are stated as (1) and (2) without state variables.

$$I_{TLT} = \begin{bmatrix} START \\ FINISH \\ Timer\_20s \\ Timer\_15s \\ Timer\_12s \end{bmatrix} = \begin{array}{c} \begin{matrix} t_0 & t_1 & t_2 & t_3 & t_4 & t_5 \end{matrix} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \quad (1)$$

$$O_{TLT} = \begin{bmatrix} T01\_V01 \\ T02\_V01 \\ T01\_L01\_V01 \\ T02\_L01\_V01 \\ L01\_V01 \\ L01\_V02 \\ L01\_V03 \\ L01\_P01 \end{bmatrix} = \begin{array}{c} \begin{matrix} t_0 & t_1 & t_2 & t_3 & t_4 & t_5 \end{matrix} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} \end{array} \quad (2)$$

A distributed control system usually contains a large number of interconnected and data-sharing devices. Redundant sequences may occur due to synchronous or asynchronous equivalent changes with a small timeshift. Therefore, dynamic time warping (DTW) is utilized to measure the similarity of collected data [23]. This method calculates an optimal match between two sequences where the mapping of the indices must be monotonically increasing within a time window. Given two time series $X = (x_1, x_2, \cdots, x_N)$ and $Y = (y_1, y_2, \cdots, y_M)$ of length $N$ and $M$, define the local cost matrix for their alignment as:

$$D \in \mathbb{R}^{N \times M}: \quad d_{i,j} = \|x_i - y_j\|, \ i \in [1, N], j \in [1, M] \quad (3)$$

The symbol $d_{i,j}$ denotes the entry in the i th row and j th column of the matrix $D$. Once the local cost matrix $D$ is built, the DTW algorithm finds the warping path which defines the correspondence of an element $x_i \in X$ to $y_j \in Y$ as $(x_i, y_j)$. $W$ is the set of warping paths that satisfy the boundary, monotonicity, and continuity conditions defined in [24]. Then, the similarity between time series $X$ and $Y$ is defined with the shortest warping path $w^*$:

$$W: \{w = w_1 w_2 \dots w_K \mid w_l = (x_{l(x)}, y_{l(y)}) \in (X, Y), l \in [1, K]\}$$

$$d_w(X, Y) = \sum_{l=1}^{K} \|w_l\|, \ \|w_l\| = d_{l(x),l(y)}$$

$$DTW(X, Y) = d_{w^*}(X, Y) = min\{d_w(X, Y), w \in W\} \quad (4)$$

The DTW(X,Y) is negatively correlated with the similarity between time series X and Y. To further identify time-shifted sequences, the other attribute Collinearity is defined by referring to the least squares method [25]:

$$X_{w^*} = [x_{w^*_{1(x)}} \; x_{w^*_{2(x)}} \; \cdots \; x_{w^*_{K(x)}}]^T$$

$$Y_{w^*} = [y_{w^*_{1(y)}} \; y_{w^*_{2(y)}} \; \cdots \; y_{w^*_{K(y)}}]^T$$

$$\overline{X}_{w^*} = [X_{w^*} \; 1]_{K\times 2}$$

$$\theta = (\overline{X}^T_{w^*}\overline{X}_{w^*})^{-1}\overline{X}^T_{w^*}Y_{w^*}$$

$$CorL(X,Y) = \frac{1}{2}(\overline{X}_{w^*}\theta - Y_{w^*})^T(\overline{X}_{w^*}\theta - Y_{w^*}) \quad (5)$$

After the similarity match, all collected data are divided into an independent set and a residual set by the decision tree model as shown in Fig. 3. The independent variable set contains no matched sequences. Each variable in the residual set can find its matched sequence in the independent variable set.

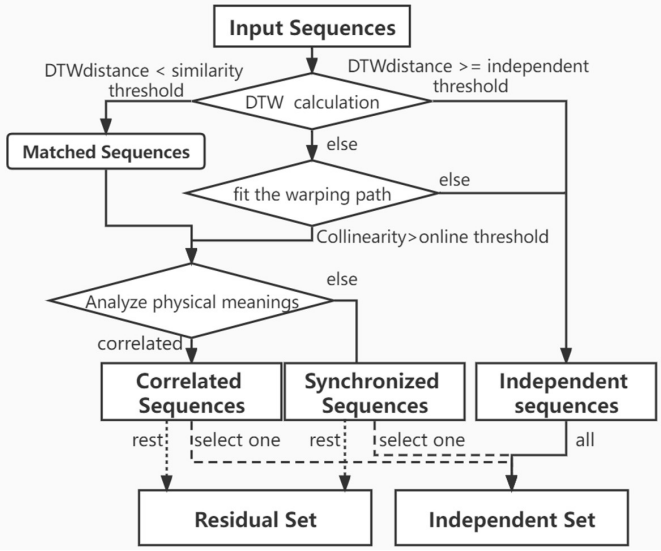


**Figure 3** Decision tree model of the similar sequences matching

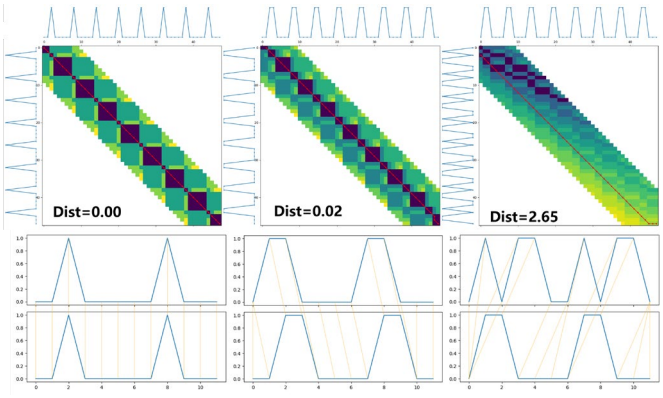


**Figure 4** DTW results of matched sequences (Left), time-shifted sequences (Middle), and independent sequences (Right).

Fig. 4 shows three matching results. Here, Manhattan distance [26] is used as the local cost measure. The smaller the distance of the indice pair is, the darker its color is in the distance matrix. White areas are indice pairs outside the time window. The red line in the middle is the warping path. In the TLT model, T01_V01 is correlated with T01_L01_V01 and synchronized with L01_V02. These three matched variables are merged and kept as T01_V in the independent set. Besides, T02_V01 is correlated with T02_L01_V01 which are merged as T02_V. Therefore, the total data volume has been reduced by 37.5% and the final independent set is as follows:

$$I = \begin{bmatrix} START \\ FINISH \\ Timer\_20s \\ Timer\_15s \\ Timer\_12s \end{bmatrix} = \begin{matrix} t_0 & t_1 & t_2 & t_3 & t_4 & t_5 \end{matrix} \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (6)$$

$$O = \begin{bmatrix} T01\_V \\ T02\_V \\ L01\_V01 \\ L01\_V03 \\ L01\_P01 \end{bmatrix} = \begin{matrix} t_0 & t_1 & t_2 & t_3 & t_4 & t_5 \end{matrix} \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} \quad (7)$$

The next step is the Hierarchical clustering. The independent set may need to be decomposed into several subsets representing multiple subsystems due to the system complexity. Similar to the I/O partitioning approach in [20], the changing pattern of variables is analyzed before the hierarchical clustering. This step adapts their strategies and includes domain-specific knowledge such as physical and process models to divide subsets. A Boolean sequence named Change Sequence is introduced to analyze the changing laws of variables in the independent set. In a distributed system, each variable has its own changing law during different process operations or actions. The Change Sequence marks the start time of each stage as 1, and others as 0. The Change Sequence of the variable v is calculated as:

$$CS_v(t) = \begin{cases} 0, & t = 0 \quad or \quad state(v(t)) == state(v(t-1)) \\ 1, & state(v(t))! = state(v(t-1)) \end{cases} \quad (8)$$

The Change Sequence describes the changing law of each independent variable. Then, the K-means time shift clustering [27] is utilized to identify their correlation. The cluster number k is equal to the number of generated FBs. It can be manually defined or automatically generated by minimizing the average within-cluster deviation. The clustered subsets of independent variables need to meet the following rules:

*Rule 1:* For any variable $v_i$ and $v_j$ $(i \neq j)$ in the same subset, there is $DTW(v_i, v_j) \geq independent\ threshold$.

*Rule 2:* For any subset $set_1$ and $set_2$, there is $set_1 \cap set_2 = \emptyset$.

*Rule 3:* For all subsets $set_1, set_2, \cdots, set_k$, there is $set_1 \cup set_2 \cup \cdots \cup set_k = I \cup O$ .In addition to data correlation analysis, some prior knowledge of the system is also taken into account. In this case, both physical and process models of such a process control system are defined in the ISA-88 standard to identify equipment groupings and describe hierarchical elements of the procedure. Fig. 5 shows the

mapping relationship of these models in the TLT-based system and subsystems are the source tank, pipeline, and target tank as the data associations defined in Table I.

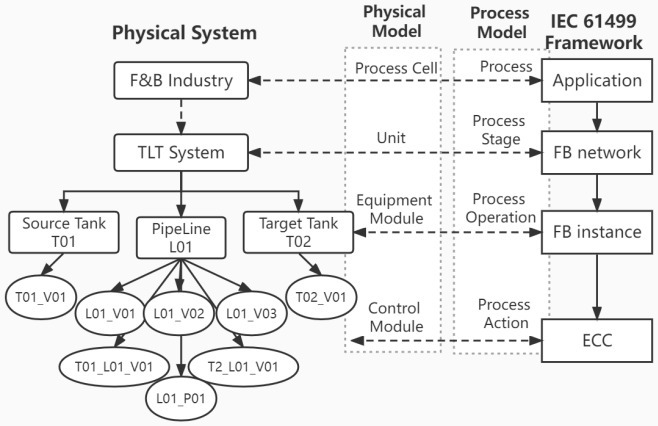


**Figure 5** The mapping relationship between the physical model (physical system) and the process model (IEC 61499 framework) in the TLT-based system

After data mining, collected I/O sequences are classified, simplified, and finally encapsulated into several subsets. Non-Boolean sequences are disassembled into multiple Boolean subsequences, and matched sequences are merged and distributed into different subsets. These steps are all to reduce the potential state space.

## 3.2 Logic Restoration

In the previous step, collected data are classified into multiple subsets. Each subset can be corresponding to one IEC 61499 FB with an ECC inside. The ECC is a deterministic finite-state machine defined in the IEC 61499. Its mathematical expression is defined as $ECC =< S, s_0, \lambda, T >$ [28]. The $S$ is a set of observed EC states and $s_0$ is the initial state. $\lambda: S \rightarrow (A_L \cup E_O)$ is an action function that represents mapping relationships between states and actions. $T: S \rightarrow 2^{(E_I \cup \{1\}) \times C(V) \times S}$ is a transition function where $E_I$ is the set of input events and $C(V)$ refers to a Boolean expression over I/O data and internal variables. The regeneration of the control logic consists of three steps: state abstraction, condition analysis, and transition mapping.

1) State abstraction: The EC state is an abstract description of the process operation in an equipment module or a unit. OS(t) is an ordered state sequence that describes system behaviors during a production cycle. Different systems vary in state descriptions, and one system may have various sets of state descriptions. An ideal state set S obey the completeness and uniqueness rules:

*Rule 1:* $\forall t \in [0, T]$, $\exists s_i \in S$ that system state $OS(t)$ could be represented by $s_i$, which is defined as $OS(t) \rightarrow s_i$.

*Rule 2:* If $s_i \in S$ and $OS(t) \rightarrow s_i$, then there is no $s_j \in S$ that $OS(t) \rightarrow s_j$.

According to ANSI/ISA S88 standard and engineer experience, a complete state set of the TLT system is $S = \{Step00, Step10, Step30, Step40, Step60, Step255\}$ as listed in Table II. Some steps are redundant for specific control modules. The system-level observed state sequence is as follows:

$$OS_{Step} = \begin{bmatrix} Step\,00 \\ Step\,10 \\ Step\,30 \\ Step\,40 \\ Step\,60 \\ Step\,255 \end{bmatrix} = \begin{array}{c} \begin{matrix} t_0 & t_1 & t_2 & t_3 & t_4 & t_5 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \quad (9)$$

TABLE II
DEFINED STATES AND CORRESPONDING PROCESS OPERATIONS OF THE TLT SYSTEM

| State Name | Process Operation | Description |
|---|---|---|
| Step00 | Wait | All devices are on standby, waiting for input signals. |
| Step10 | Fill | Connect the source tank and the pipeline, then fill the pipeline with raw materials. |
| Step30 | Production | Connect the source tank and the target tank, then pipe the raw materials. |
| Step40 | Wash | Disconnect all tanks from the pipeline, then flush the pipeline with clean water. |
| Step60 | Empty | Close the water valve while keep the drain valve open to drain the pipeline. |
| Step255 | Stop | Detect the status of all devices, trigger the Complete flag and return to Step00. |

2) Condition analysis: The condition of the EC transition consists of a trigger event with/without a guard condition. $t_{ij}: s_i \xrightarrow{(ie_{ij}) \cdot (c_{ij})} s_j$ is a transition template in which the object leaves the current state s_i and enters the next state s_j when the input event ie_ij reaches and the guard condition c_ij is satisfied. The trigger event ie is either a particular input event or 1 (any event). The guard condition c is a Boolean expression that applies logical operators to logical or arithmetic expressions over I/O and internal variables.

If each transition has its input signal (condition satisfied or event triggered), the observed system is considered as controllable. If each transition has its output signal (output variable changed or output event triggered), the observed system is considered as measurable. I/O mismatches may occur due to sampling truncation, invalid input, internal behavior, etc., and can be circumvented by repeated sampling. This paper focuses on the ideal sampling of controllable objective systems. The mapping relationships between the observed state sequence OS(t) and input/output event sequence IE(t)/ OE(t) are given as:

$$\frac{IE}{OS} = \left[\frac{IE(0)}{OS(0)}\right], \left[\frac{IE(1)}{OS(1)}\right], \left[\frac{IE(2)}{OS(2)}\right], \ldots \quad (10)$$

$$\frac{OS}{OE} = \left[\frac{OS(0)}{OE(0)}\right], \left[\frac{OS(1)}{OE(1)}\right], \left[\frac{OS(2)}{OE(2)}\right], \ldots \quad (11)$$

The TLT model has an input event set $E_I = \{INIT, REQ, START, FINISH, STOP\}$ which contains collected I/O data $\{START, FINISH\}$, function block basic event $\{INIT, REQ\}$, and a hidden event *STOP*. The guard condition set $C(V)$ contains only one expression $(T02.Level.add == Amount)$ . Therefore, an ECC template of the general pipeline transportation process is generated based on domain-specific knowledge. This template listed all process operations from $Step00$ to $Step250$ as EC states. Each state has one undefined normal execution action and one Error action designed to throw a fault. No I/O variables but some abstract conditional directed arrows are involved to indicate the execution order.

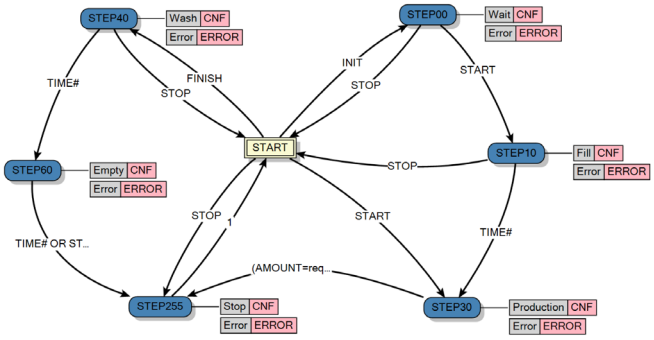


**Figure 6** ECC template of general pipeline transportation process

As shown in Fig. 6, Step00 will jump to Step10 when the input event *START* is triggered after initialization is completed. Secondly, Step10 will jump to Step30 after a set time delay. When the raw material reaches the set value, Step30 will jump to the stop state Step255, waiting for the next instruction. If the input signal is *START*, repeat Step30; if the input signal is *FINISH*, go to Step40. Step40 transitions to Step60 and then Step 255 all after a set time delay. Error responses are not covered in this state machine. Here, the generated ECC only gives feedback based on the status of each step. The current state will return to the initial state and throw an output event *ERROR* when a fault occurs.

3) Transition mapping: The generated ECC template needs to be parameterized and mapped before deployment. Here, global and internal variables in the original PLC code are replaced with a pair of input/output variables. *Algorithm 1* restores the missing parameters in the template and optimizes the final ECC by analyzing I/O and state sequences. $s_i$ and $s_j$ are pre-defined EC states in the set $S$. The action function $\lambda$ is generated by mapping the destination state with synchronized outputs. The state transition $t_{ij}$ is generated by applying the operator *AND* to synchronously triggered input events and guard conditions. Transition conditions of the recurring state transition are connected by the operator *OR*. Then, an ECC is generated by merging all state transitions and action execution.

**Algorithm 1** ECC generation algorithm

**Input:** state set $S$:array[1..N], input event sequence $E_I(t)$:array[1..M], guard condition set $C$:array[1..K], output sequence $O(t)$:array[1..M] observed state sequence OS:array[1..P]
**Output**: transition function $T$, action function $\lambda$

$\lambda: s_0 \rightarrow O(0)$
**while** t < $P$ **do**
  //Detect a change in the observed state sequence
  **if** $OS[t] \neq OS[t-1]$ **then**
    //Find matched elements in the state set
    find $s_i \in S$ that $s_i == OS[t-1]$
    find $s_j \in S$ that $s_j == OS[t]$
    //Map outputs to the new state
    $\lambda: s_j \rightarrow O(t)$
    //For new transition, initialize its input event as 1
    **if** $ie_{ij} \notin E_I$ **then**
      $ie_{ij} := 1$
    **end if**
    //Modify the input event
    **for** $ie_m \in E_I$ **do**
      //Add the triggered input event with AND
      **if** $ie_m(t)$ **then**
        $ie_{ij} \leftarrow ie_{ij} \wedge ie_m$
      **end if**
    **end for**
    //For new transition, initialize its guard condition as 1
    **if** $c_{ij} \notin C$ **then**
      $c_{ij} := 1$
    **end if**
    //Modify the guard condition
    **for** $c_k \in C$ **do**
      //Add the satisfied input event with AND
      **if** $c_k(t)$ **then**
        $c_{ij} \leftarrow c_{ij} \wedge \{c_k \cdot c_k(t)\}$
      **end if**
    **end for**
    //Add or update $t_{ij}$ in the transition function $T$
    **if** $t_{ij} \notin T$ **then**
      $T \leftarrow T \wedge t_{ij}$ that $t_{ij}: s_i \xrightarrow{(ie_{ij}) \bullet (c_{ij})} s_j$
    **else**
      update $t_{ij} \leftarrow t_{ij} \vee \{(ie_{ij}) \bullet (c_{ij})\}$
    **end if**
  **end if**
**end while**

It should be noted that a valid ECC has only one active state at the same time. Therefore, concurrent transitions need to be prevented. With the priority of EC transitions introduced in the IEC 61499 second edition, each EC transition will be assigned with a different priority. If more than one EC transition condition is met, the EC transition with the highest priority will be activated. If no priority options are provided in the IEC 61499 platform, manual modification of transition conditions is required. The final ECC satisfies the observed state sequence and outputs the same results under the same input as the original system.

The introduction of domain-specific templates makes the identification problem nondeterministic polynomial. Fig. 7 and Fig. 8 show the generated ECC of the source and destination tank FB. The ECCs of the two FBs are very

similar except the STEP20 in the source tank FB is replaced with the STEP30 in the destination FB. In general, 7 transition parameters (mostly the time durations) in the original template are restored and 4 instance states are removed.

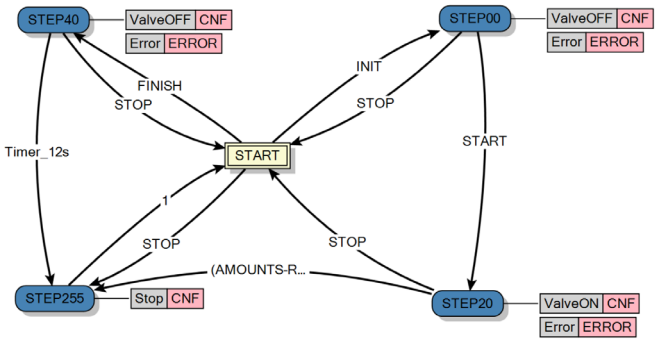


**Figure 7** Generated ECC of the source tank in the TLT system

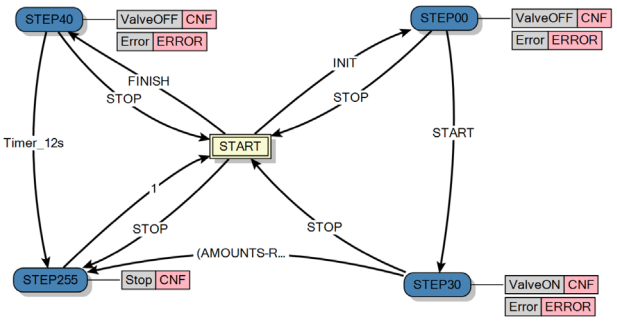


**Figure 8** Generated ECC of the destination tank in the TLT system

## 3.3 System Model Recovery

The final step is to recover the entire system model which can be divided into three steps: interface recovery, instantiating, and connection establishment.

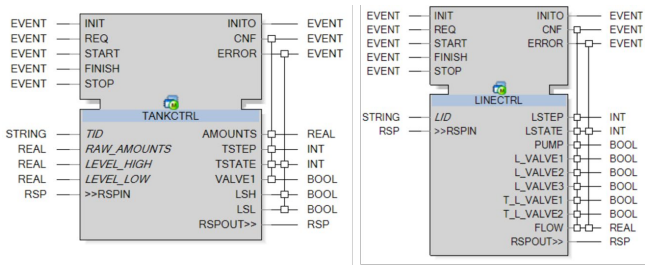


**Figure 9** Designed interface of the TANKCTRL function block (left) and the LINECTRL function block (right)

1) Interface recovery: A function block interface $I =< E_I, V_I, \alpha_I, E_O, V_O, \alpha_O >$ is defined as a tuple where $E_I$, $V_I$, $E_O$ and $V_O$ are finite sets of I/O events and data, while $\alpha_I \subseteq E_I \times V_I$ and $\alpha_O \subseteq E_O \times V_O$ are the sets of input and output associations [27]. The reconstruction of the external FB interface mainly depends on the variables required by internal functions in each mapped device. For the TLT-based pattern, equipment modules are mapped into two function block types: *TANKCTRL* and *LINECTRL*. Fig.9 shows the designed interface of these two FBs.

$TANKCTRL.TID$ and $LINECTRL.LID$ identify the equipment modules after instantiating. *RSP* is a combined input signal structure, including all user inputs on the HMI. Other variables are less relevant to the production process but highly related to system monitoring and safety.

2) Instantiating: The purpose of instantiation is to instantiate and reconstruct (if necessary) the generated function block types as FB instances. The key to instantiation is matching the existing function block types to specific system behaviors and then allocating them to specified devices.

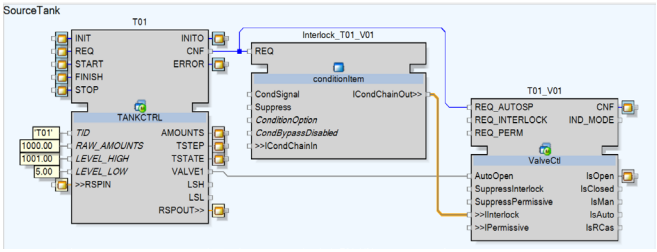


**Figure 10** The FB sub-network of the source tank in the TLT model

Fig. 10 shows the sub-network of the source tank and downstream devices in the TLT model. The valve control module *ValveCtl* and the motor control module *Motor1S1D* are applied here to handle I/O modules.

3) Connection establishment: A FB network consists of several FB instances interconnected by event and data [6]. The data connections aim to exchange information, while the event connections aim to propagate control signals. Events are connected according to the system structure and execution logic, while data are connected according to the paired sequences in the previous similarity match.

The IEC 61499 standard provides a flexible deployment plan in that FBs are mapped to various resources at the runtime stage. These resources could be controllers on site, edge gateways, or even cloud servers. With flexible deployment ability provided, a cloud-edge collaborated hierarchical application architecture is proposed.

As illustrated in Fig.11, the reconstructed FB application network contains three hierarchical layers: central control, equipment modules, and control modules. The central control monitors system states and handles errors. Equipment modules get the upstream outputs and send control signals to control modules mapped to downstream devices like valves

and motors. Two reusable FB types (*TANKCTRL* and *LINECTRL*) are generated with built-in ECC templates suitable for most pipeline transportation scenarios. Users only need to modify state outputs and device mapping for instantiation.

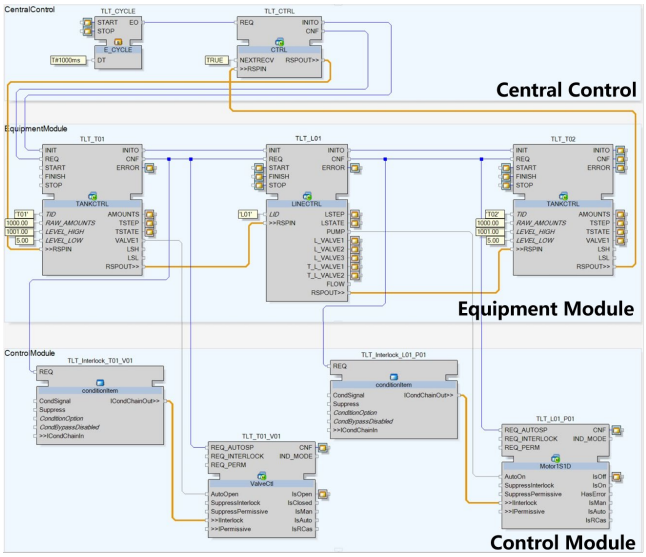


**Figure 11** The reconstructed FB network of the TLT-based system

Overall, a deployable FB network is reconstructed semi-automatically. Raw operation data are processed and clustered automatically. ECCs of these independent sets are reconstructed via model-driven transformation and data-driven parameter recovery. The third module designs the external interface and instantiates FB types manually. Finally, FB instances are interconnected to a complete network following the semantics of the system behavior.

## 4. Case Study and Implementation

A case study of the milk canning system is used to prove the proposed method as shown in Fig.12. The milk canning system is a typical process control system, which contains all essential components, including valves, tanks, and pumps, that exists for all types of process control systems. This system contains eight equipment modules (two source tanks R1BA and R1BB, two pipelines A1L01 and A1L02, four target tanks TA11-14) and 22 control modules (twelve one-way valves, eight two-way valves, two motors). Raw materials can be transported from any milk cart (R1BA or R1BB) to any milk tank (TA11-14) via any combination of pipeline (TA11-14_V01 to TA11-14_V02). For example, the original milk can be filled from the R1BA via the pipeline A1L01_V01 and add cold water by A1L01_V02 and then flow into the milk tank TA11 via the value TA11_A1L01_V01 and TA11_V01. The chocolate powder can be added and mixed with the original milk, and finally sent to the filling lines via the pipeline A1L01.

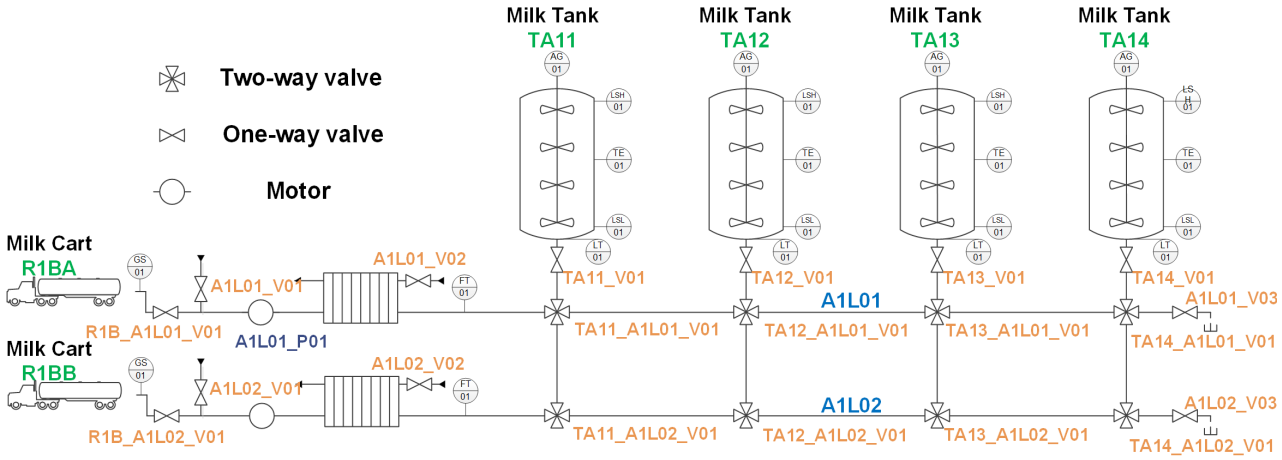


**Figure 12** Process & Instrument Diagram of the milk canning system

Fig.13 shows the entire cloud-edge collaborated architecture. More than 10,000 pieces of production data and HMI signals from 8 production processes are collected from Schneider Electric Modicon M580 PLCs to the cloud platform. The production data contain sensor signals and control signals of all devices listed in Fig.12. The HMI signals are the same as the TLT system.

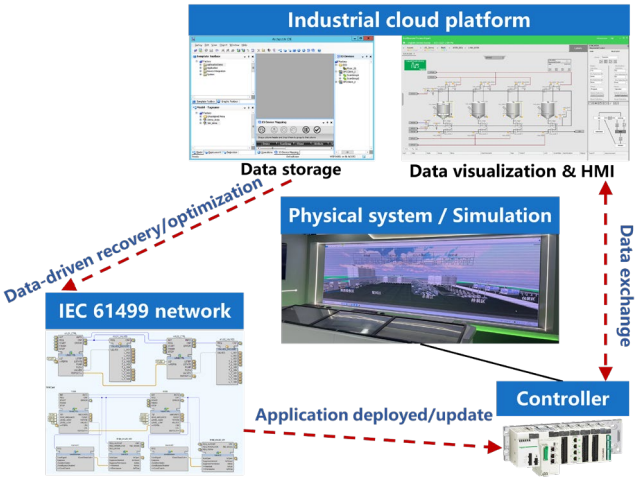


**Figure 13** Cloud-based architecture for application and validation

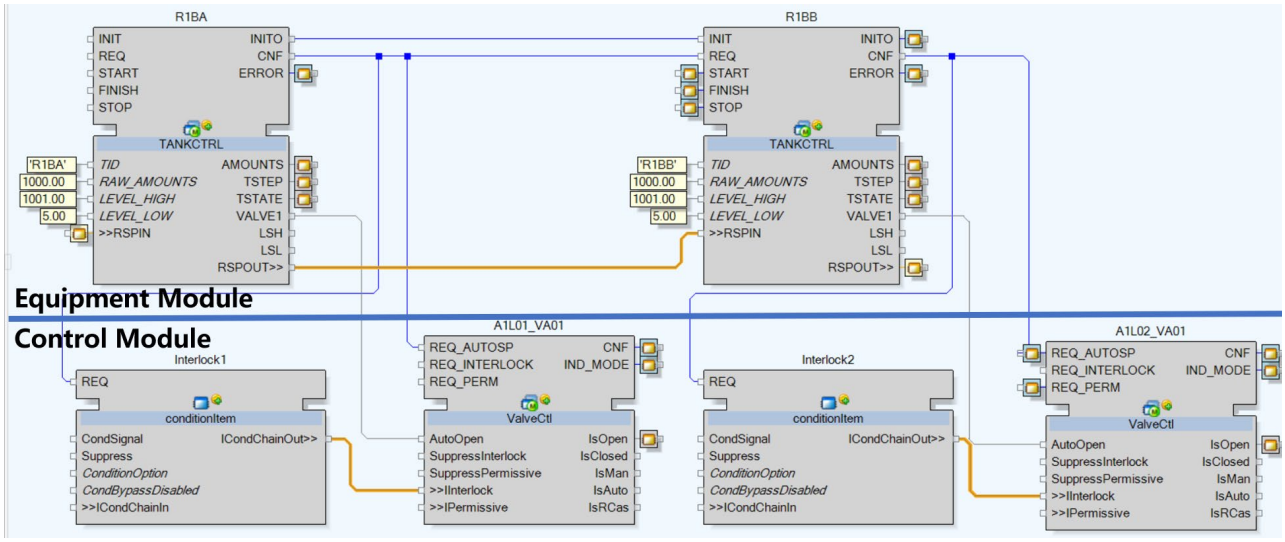


**Figure 14(a)** The FB Network of the milk cart control

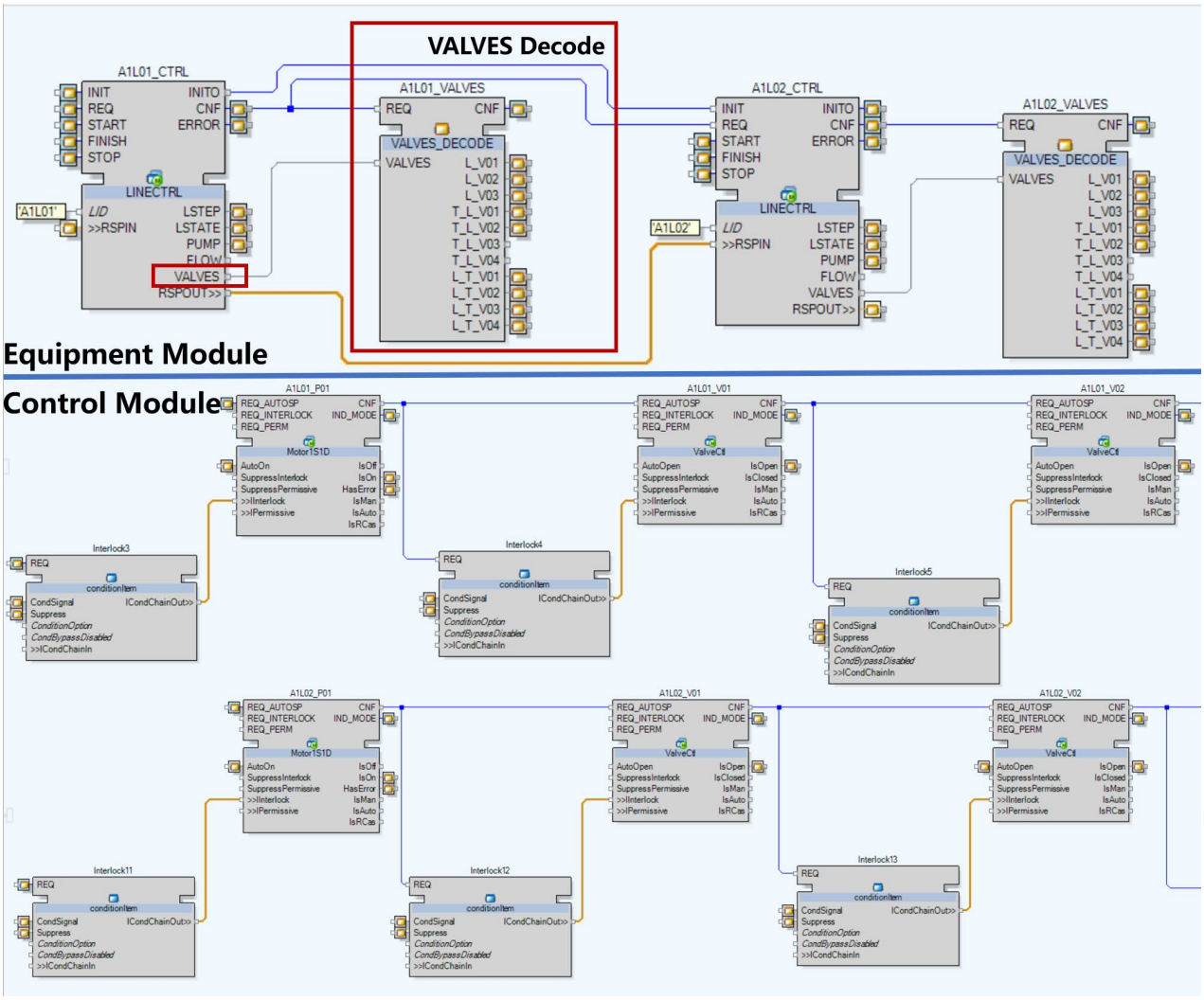


**Figure 14(b)** Part of the FB sub-network of the pipeline control

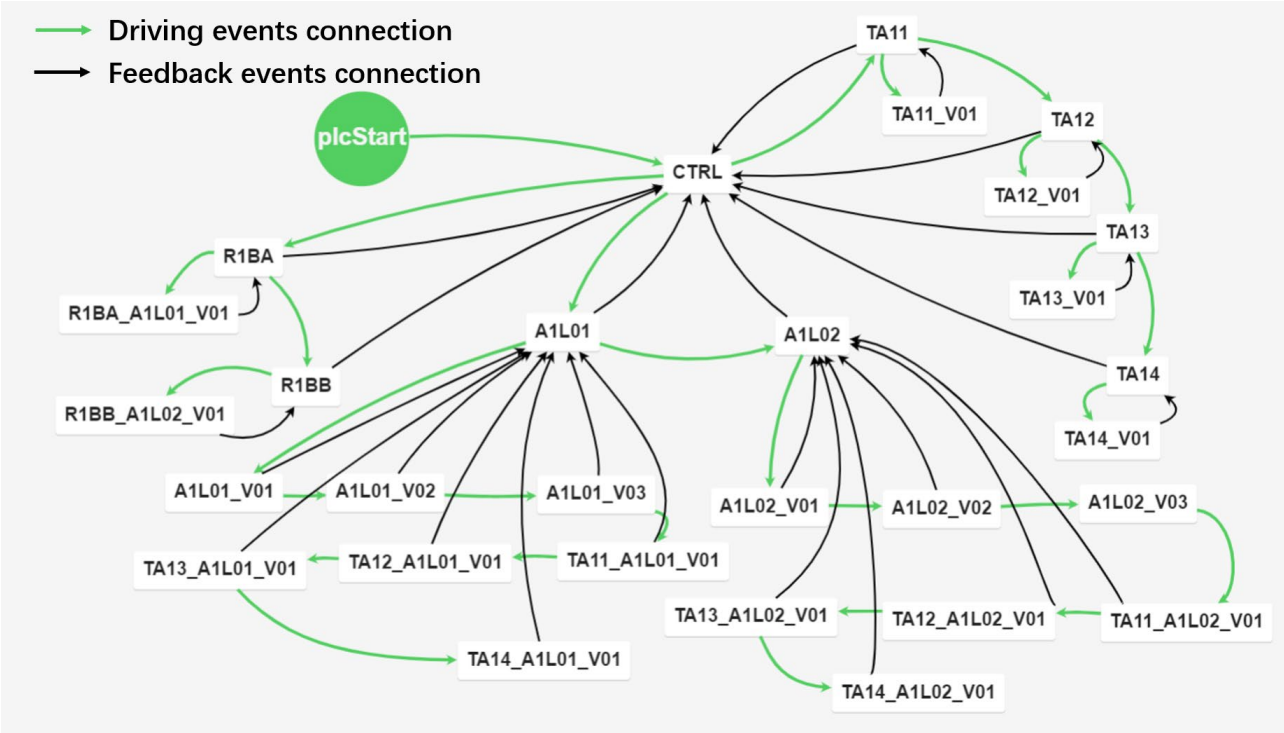


**Figure 15** Control Flow Diagram of the raw milk canning system

pairs of correlated sequences are identified. The control signal *R1B_A1L01_V01* of the valve at the junction of the milk cart and the pipeline A1L01 is matched with *A1L01_V01* which controls the ice water inlet valve. Similarly, *R1B_A1L02_V01* is matched with *A1L02_V01* in the pipeline A1L02. Therefore, independent variables are finally clustered into eight equipment modules.

This system acts as a combination of multiple freely matched TLT systems. Two milk cart modules and four milk tank modules are instantiated as *TANKCTRL* instances. Two pipelines are instantiated as *LINECTRL* instances in which valve control outputs are combined into a defined structure *VALVES* and decoded by a basic FB *VALVES_DECODE*. Fig.14a and Fig.14b show the network of the milk cart control and the pipeline control.

The hierarchical structure of this system is similar to the TLT system. The only difference is that the equipment modules are divided into three sub-networks: milkCart, pipeline, and milkTank to facilitate management. These sub-networks are cascaded, while their internal function blocks are controlled in a distributed manner. The control flow diagram (a simplified version of the FB network) of the entire system is given in Fig. 15. The logic inside FBs are implemented using the IEC 61131-3 ST language.

Qualitative and quantitative analyses of the generated results are presented in Fig. 16 and Table III. Indicators of Timeliness and Accuracy are applied to validate that requirements are satisfied on time and correctly. Timeliness calculates the sum of the time difference when the milk tank reaches the required liquid level (marked as red rectangles in

TABLE III

GENERATION RESULTS AND OUTPUT VALIDATION

| | Collected Data | | | | Generated Model | | | Validation | |
|---|---|---|---|---|---|---|---|---|---|
| | T-L-T num | Transport times | I/O num | time/s | I/O matched rate | FB num | state num | Timeliness | Accuracy |
| sample case | 1-1-1 | 1 | 19 | 157 | 100% | 21 | 14 | +390ms | 100% |
| | 1-1-2 | 2 | 22 | 268 | 100% | 26 | 18 | +320ms | 100% |
| | 2-1-1 | 2 | 22 | 275 | 100% | 26 | 18 | +270ms | 100% |
| | 2-2-1 | 4 | 28 | 381 | 100% | 34 | 24 | -50ms | 100% |
| | 2-2-4 | 16 | 40 | 990 | 100% | 52 | 36 | -310ms | 100% |
| random test | 2-2-4 | 3 | 22 | 342 | - | - | - | -110ms | 100% |
| | 2-2-4 | 6 | 24 | 503 | - | - | - | -200ms | 98% |
| | 2-2-4 | 12 | 40 | 637 | - | - | - | -280ms | 95% |
| | 2-2-4 | 20 | 40 | 811 | - | - | - | -440ms | 95% |

Fig.17), while Accuracy compares the output similarity of each process operation. Repeated simulation results show that the generated FB network completed the transmission tasks 100% as the original system within the required time. The restoration rate of input conditions other than the collected data drops to 95% due to failure to detect wrong input.

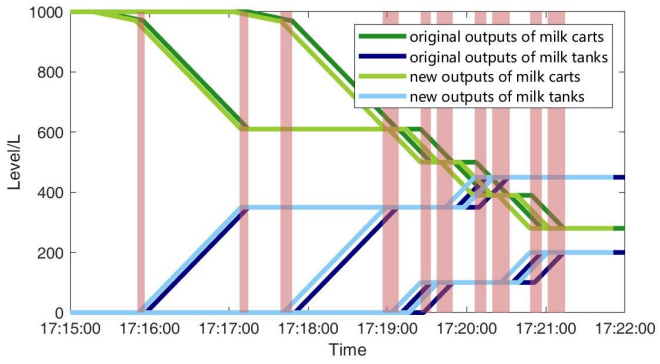


**Figure 16** Comparison between the outputs of the generated FB networks and the original control code for a certain period of time

The accuracy results proved that the proposed framework can be applied to any process control automation system with only Boolean inputs and outputs. Although the accuracy cannot achieve 100%, with IEC 61499 FBs, the missing part can be manually adjusted.

## 5. Conclusions and Future Work

Migration from legacy automation systems is quite challenging due to hidden complexities in internal structures. This paper proposed a data-driven recovery method that is suited for refactoring the control logic of distributed legacy systems with linear process flow with independent workstations. The goal of restoration is to generate a function block network that executes the same actions and outputs the same results under the same inputs. A generic TLT pattern is proposed to model process control systems. This approach not only recovers state machine-based reusable software components for process control industries but also generates a complete system application based on the IEC 61499 which can be directly deployed. The proposed approach provides a generic approach to migrate legacy automation systems without source code available to avoid re-implementing the entire logic again. Also, with the IEC 61499 FB models, the recovered model can be distributed across multiple devices without modifying any logic that is a hassle in multiple PLC programming.

To achieve fully automatic migration from legacy control systems, several improvements could be investigated. Firstly, the proposed logic restoration method still highly relies on domain-specific knowledge. Unseen input data need to be inferred based on empirical requirements. Common methods for discrete manufacturing and process automation will be investigated to reduce the dependency on domain models. This approach needs to be extended to achieve a fully automatic process for migrating both process control and discrete manufacturing systems. Secondly, automatic systematic validation and formal verification algorithms must be introduced to ensure system safety. Finally, the recovery process can be linked with the automatic code generation process to achieve a closed-loop framework that can automatically optimize behavior from operation data.

### Acknowledgements.

This research work is sponsored by the National Natural Science Foundation of China, Project No. 92467301.

## References

[1] A. Colombo, S. Karnouskos, and T. Bangemann, “Towards the next generation of industrial cyber-physical systems”, Industrial cloud-based cyber-physical systems. Springer, pp. 1-22, 2014.

[2] J. O. Ringert, B. Rumpe, C. Schulze and A. Wortmann, “Teaching agile model-driven engineering for cyber-physical systems,” 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET), Buenos Aires, pp. 127-136, 2017.

[3] M. A. Garzo´n, H. Aljamaan and T. C. Lethbridge, “Umple: A framework for Model Driven Development of Object-Oriented Systems,” 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Montreal, QC, 2015, pp. 494-498.

[4] F. Ding, J. Simpson and Y. Zhang, “A Transparent Translation from Legacy System Model into Common Information Model,” 2018 IEEE/PES Transmission and Distribution Conference and Exposition (T&D), Denver, CO, pp. 1-5, 2018.

[5] T. Dharmawan and S. Rochimah, “Systematic literature review: Model refactoring,” 2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT), Kuta Bali, pp. 1-5, 2017.

[6] IEC 61499-1: Function Blocks Part 1: Architecture, International Electrotechnical Commission, 2012.

[7] G. Lyu and R. W. Brennan, “Towards IEC 61499-Based Distributed Intelligent Automation: A Literature Review,” in IEEE Transactions on Industrial Informatics, vol. 17, no. 4, pp. 2295-2306, April 2021.

[8] IEC 61131–3: Programmable controllers Part, International Electrotechnical Commission, 2013.

[9] A. Keliris and M. Maniatakos, “ICSREF: A Framework for Automated Reverse Engineering of Industrial Control Systems Binaries”, Network and Distributed Systems Security Symposium, pp 1-16, 2018.

[10] R. Keller, R. Schauer, S. Robitaille and P. Pagé, “Pattern-based reverse-engineering of design components”, In Proceedings of the 21st international conference on Software engineering, pp. 226-235, 1999.

[11] P. Sun, L. Garcia and S. Zonouz, “Tell Me More Than Just Assemble! Reversing Cyber-physical Execution Semantics of Embedded IoT Controller Software Binaries”, 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp 349-361, 2019.

[12] S. Patil, D. Drozdov and V. Vyatkin, “Adapting Software Design Patterns To Develop Reusable IEC 61499 Function Block Applications,” 2018 IEEE 16th International

Conference on Industrial Informatics (INDIN), Porto, pp. 725-732, 2018.
[13] L. H. Yoong, P. S. Roop, Z. E. Bhatti, and M. M. Kuo, Model-driven Design Using Iec 61499: A Synchronous Approach for Embedded and Automation Systems. Springer, 2014.
[14] F. Andrén, T. Strasser, and W. Kastner, "Engineering Smart Grids: Applying Model-Driven Development from Use Case Design to Deployment," Energies, vol. 10, no. 3, p. 374, Mar. 2017
[15] M. Wenger and A. Zoitl, "Re-use of IEC 61131-3 Structured Text for IEC 61499," 2012 IEEE International Conference on Industrial Technology, Athens, pp. 78-83, 2012.
[16] P. Gsellmann, M. Melik-Merkumians, A. Zoitl and G. Schitter, "A Novel Approach for Integrating IEC 61131-3 Engineering and Execution into IEC 61499," in IEEE Transactions on Industrial Informatics, Vol. 17, No. 8, pp. 5411-5418, 2020.
[17] D. Chivilikhin, S. Patil, K. Chukharev, A. Cordonnier and V. Vyatkin, "Automatic State Machine Reconstruction From Legacy Programmable Logic Controller Using Data Collection and SAT Solver," in IEEE Transactions on Industrial Informatics, vol. 16, no. 12, pp. 7821-7831, Dec. 2020.
[18] J. Ladiges, C. Haubeck, A. Fay and W. Lamersdorf, "Learning Behaviour Models of Discrete Event Production Systems from Observing Input/Output Signals", International Federation of Automatic Control, 48(3), pp 1565-1572, 2015.
[19] J. Ladiges, A. Fulber, E. Arroyo, A. Fay, C. Haubeck and W. Lambersdorf, "Learning Material Flow Models for Manufacturing Plants from Data Traces", IEEE 13th International Conference on Industrial Informatics, pp 294-301, 2015.
[20] A. P. Estrada-Vargas, E. López-Mellado and J. Lesage, "A Black-Box Identification Method for Automated Discrete-Event Systems," in IEEE Transactions on Automation Science and Engineering, vol. 14, no. 3, pp. 1321-1336, July 2017.
[21] J. Saives, G. Faraut and J. Lesage, "Automated Partitioning of Concurrent Discrete-Event Systems for Distributed Behavioral Identification," in IEEE Transactions on Automation Science and Engineering, vol. 15, no. 2, pp. 832-841, April 2018.
[22] D. Ivanova, G. Frey and I. Batchkova, "Intelligent component based batch control using IEC61499 and ANSI/ISA S88," 2008 4th International IEEE Conference Intelligent Systems, 2008, pp. 4-44-4-49.
[23] P. Senin, "Dynamic time warping algorithm review," Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA, vol. 855, no. 1-23, p. 40, 2008.
[24] S. Seto, W. Zhang and Y. Zhou, "Multivariate Time Series Classification Using Dynamic Time Warping Template Selection for Human Activity Recognition," 2015 IEEE Symposium Series on Computational Intelligence, pp. 1399-1406, 2015.
[25] R. V. Lenth, "Least-squares means: the r package lsmeans," Journal of statistical software, vol. 69, pp. 1–33, 2016.
[26] W. Chiu, G. G. Yen and T. Juan, "Minimum Manhattan Distance Approach to Multiple Criteria Decision Making in Multiobjective Optimization Problems," in IEEE Transactions on Evolutionary Computation, vol. 20, no. 6, pp. 972-985, Dec. 2016.
[27] M. Roux, "A Comparative Study of Divisive and Agglomerative Hierarchical Clustering Algorithms", Journal of Classification, vol. 35, No. 2, pp.345-366, 2018.
[28] R. Sinha, P. S. Roop, G. Shaw, Z. Salcic and M. M. Y. Kuo, "Hierarchical and Concurrent ECCs for IEC 61499 Function Blocks," in IEEE Transactions on Industrial Informatics, vol. 12, no. 1, pp. 59-68, Feb. 2016.