Journal of Electronic Research and Application, 2025, Volume 9, Issue 3
http://ojs.bbwpublisher.com/index.php/JERA
ISSN Online: 2208-3510
ISSN Print: 2208-3502

AI-Driven Implementation of Universal Control

Zhi Tang*

Sichuan Coal Industry Group Co., LTD, Dazhou 635000, Sichuan, China

**Author to whom correspondence should be addressed.*

Abstract: Control serves the process technology. Therefore, the optimal implementers of process control should be process engineers rather than control engineers. While process engineers best understand the control requirements of the technology, the inherent complexity of control systems has rendered many process engineers incapable of accomplishing these control tasks [1]. The greatest truths are the simplest, from abacus to calculator, from DOS to Windows, each technological revolution has been driven by application-oriented simplification. Can integrated automation in process industries also achieve such extreme simplicity and rapid implementation? This is precisely what this paper aims to explore.

Keywords: Data-driven; Observer; Organizer; Parser

Online publication: 29 May, 2025

1. Introduction

Like many control tasks, the diversity of user process requirements dictates that control programs cannot be pre-configured. Control engineers must implement them through individual programming, an inherently tedious process [1]. To simplify this workflow, extensive research and efforts have been undertaken. For instance, on the hardware side, universal PLC controllers have been developed. On the software side, standardized programming languages such as IEC 61131-3 and IEC 61499 have been defined to improve usability. As process requirements evolve, technologies like flexible control, distributed control, and open control have also been explored. However, many practical challenges remain unresolved or difficult to address effectively.

To tackle these issues, a novel control implementation method is proposed: Data-driven + Observer pattern. The Observer leverages data-driven characteristics to provide process data required for control, centered on the outcome of user-defined process requirements. This enables the Parser within the controller to directly, simply, quickly, and transparently execute user-specific control tasks [2]. Since these processes are automated by intelligent components in FlashControl, the system conceals extensive complexity, making control implementation remarkably straightforward. Consequently, the system is named FlashControl, an abbreviation of 'Instant Control.'

2. Data-driven pattern

Traditionally, varying user process requirements have been addressed by modifying code to adapt to specific needs. This approach necessitates code compilation, linking, and debugging, and requires control engineers to repetitively perform extensive, labor-intensive tasks. From a macro perspective, this workflow is highly inefficient.

A program comprises two components: code and data. Shifting perspectives, what if the code remains unchanged while control data dynamically adapts to different user processes? Could this still fulfill user-specific control requirements? This is precisely the Data-Driven Pattern explored in this paper.

By leveraging data, which requires no compilation or linking, and enabling AI-powered components within FlashControl to autonomously organize, optimize, and allocate control data, the implementation process is significantly simplified while maintaining universality. The framework is illustrated below:

From **Figure 1**, the data-driven approach serves as the cornerstone of minimalist control and allows AI to achieve universal control, owing to data's inherent flexibility and adaptability when contrasted with rigid code.

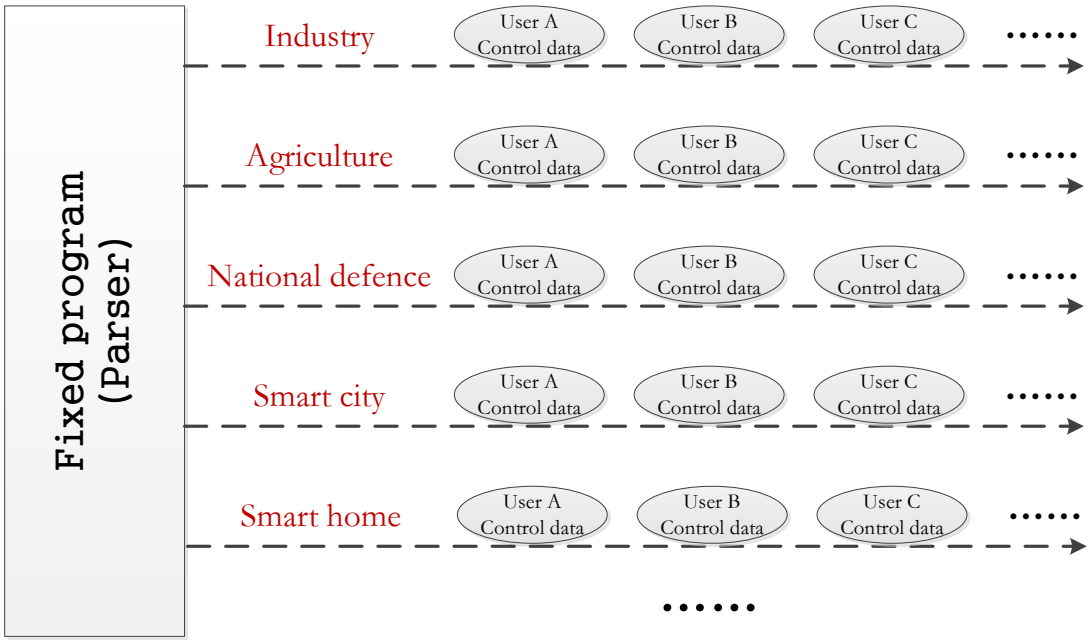

Figure 1. Data-driven mode diagram

3. Observer controlling data collection

The Observer, a core intelligent component of FlashControl, functions by:

(1) Using software/hardware modules as building blocks;

(2) Following the hardware resource topology as pathways;

(3) Basing itself on process logic topology to observe users' control requirements;

(4) Defining runtime conditions and data flow paths for process controls and components;

(5) Capturing attributes of control data and interfaces... all recorded in the Connection Database [2].

Since the Observer primarily handles relationships between system resources, this relationship-recording database is termed the Connection Database.

The Observer pattern represents a significant innovation distinct from traditional control methods. It evolves text-based logic descriptions into graphical process logic descriptions that are intuitive for process engineers.

Rather than focusing on implementation details, it observes the topological relationships, connections, structures, and data attributes between hardware modules, software modules, and their interactions—all from the perspective of the user's engineering system [3]. This process is autonomously executed by FlashControl's intelligent components, enabling effortless and rapid control implementation.

The data structures and relationships captured in the Connection Database correspond to the content control engineers traditionally program into control code. The key distinction lies in methodology: conventional approaches describe control through code, while the Observer describes control through data. The Observer itself does not execute control but instead transfers observed data to the Organizer for further processing.

The Observer pattern plays a vital role in achieving process flexibility:

It leverages the adaptability of data to accommodate process diversity, rather than relying on rigid one-to-one programming to combat variability—an effective pathway for flexible process implementation [4].

Figure 2 shows the diagram of the Connection Database generated by the Observer.

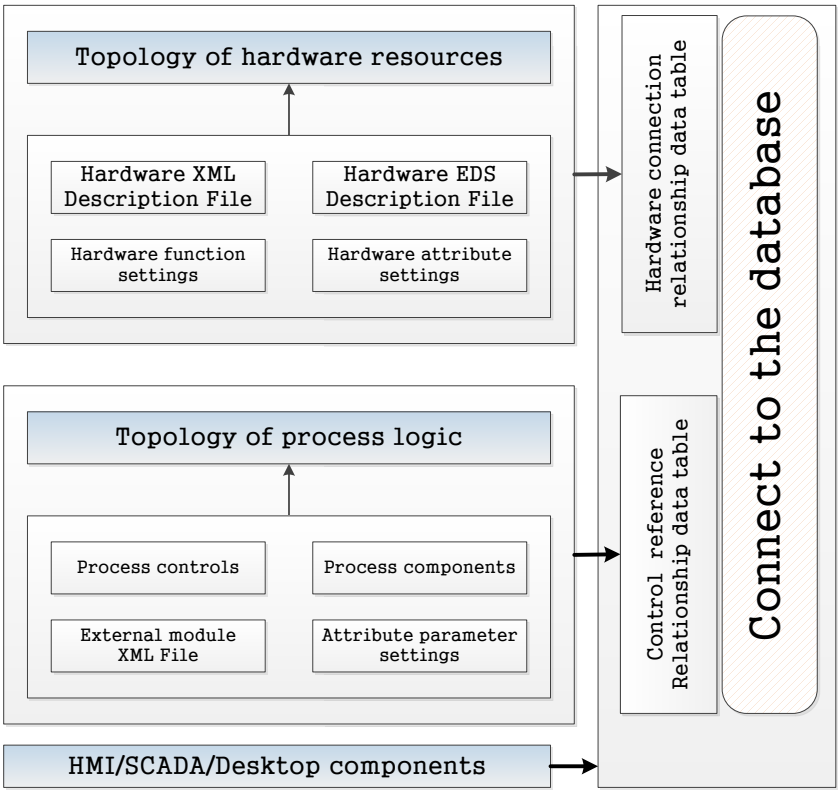

Figure 2. Diagram of Observer data collection

3.1. Process (logic) topology—intuitive user requirement description

How we describe a problem is critically important. Clear, intuitive descriptions **are** the starting point for simplification, and this principle applies equally to capturing user requirements for industrial processes. A minimalist implementation demands a framework that prioritizes process engineers rather than control engineers in understanding requirements, which is FlashControl's primary objective [2]. While standards like IEC 61131-3 strive for simplicity, they remain bound to controller-specific programming. In contrast, Process Topology represents a top-level minimalist design tailored for process engineers, focusing on the user's industrial system as the object of analysis.

Unlike conventional methods that rely on detailing control procedures to achieve objectives, Process Topology ignores implementation minutiae and directly describes the desired outcomes of process control. FlashControl then intelligently deduces the procedural data required to achieve these outcomes, a core philosophy enabling minimalist control implementation [2]. Crucially, whether FlashControl can accurately analyze and derive these details from acquired information is the central focus of this paper.

According to Figure 3, it can be observed that a process topology consists of functional modules and process control elements that interconnect these modules. A user system can be composed of multiple process topologies, with its control architecture [illegible] into two layers: (1) the core layer; (2) the connection layer. The architecture of a process topology is essentially independent of the complexity of user projects.

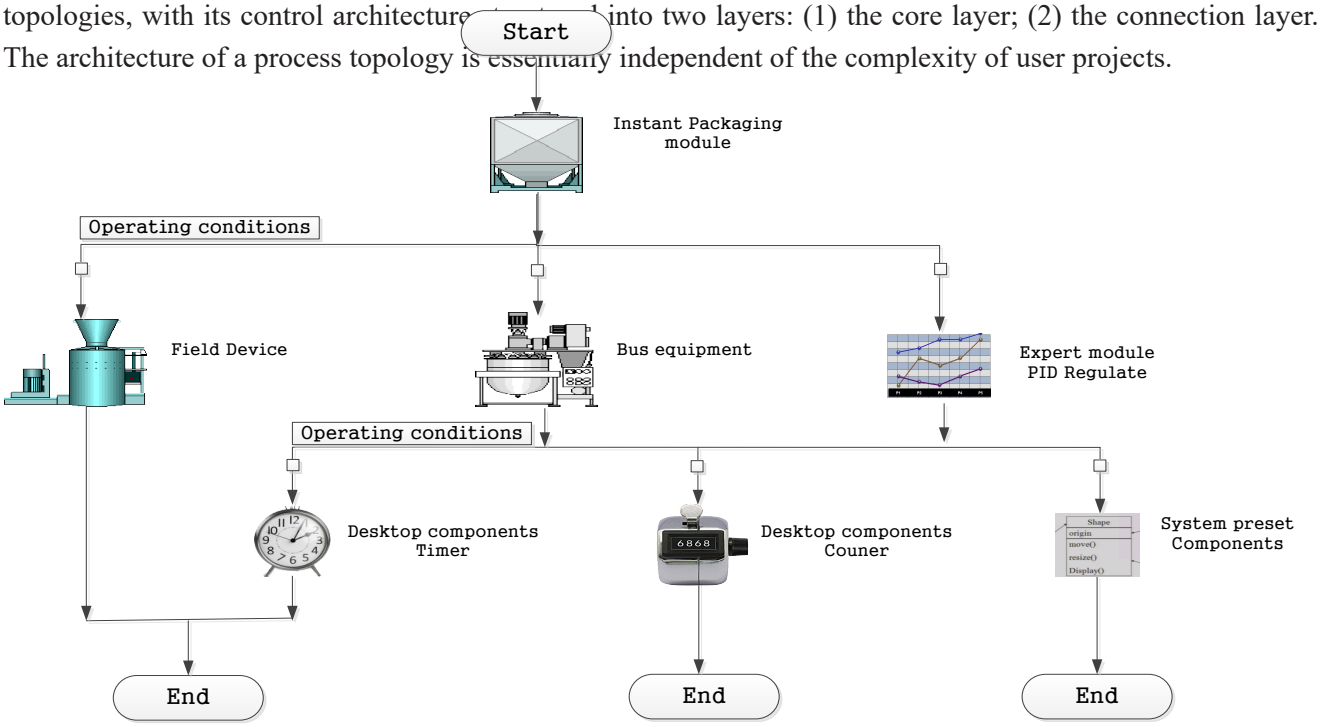

Figure 3. Schematic diagram of engineering topology

3.1.1. Core layer

The core layer is composed of various software functional modules. These functional modules, also referred to as Function Block Diagrams (FBDs), primarily originate from:

(1) Hardware-associated functional modules integrated with intelligent hardware;

(2) Generic functional modules pre-installed in FlashControl;

(3) Customized functional modules encapsulated by users as needed;

(4) Specialized functional modules contributed by industry experts.

The mapping of these functional modules is collectively termed process components in a process topology. Each process component is associated with a functional module and corresponds to a background data block, which records the conditional data and attributes required for the module's operation [3]. Process components prepare runtime data for functional modules through the UI layer.

A large pre-existing library of selectable functional modules is critical to simplifying control implementation. However, unless the source code is open, functional modules are almost always proprietary. The public sharing of functional modules holds profound significance: it enables the high-level accumulation and widespread application of expert knowledge, avoiding inefficient duplication. Achieving this goal requires addressing two

key challenges: motivating industry experts to contribute knowledge modules and enabling the transplant-free application of functional modules.

3.1.1.1. Motivation for FBD openness—black box

Expert knowledge modules may encapsulate decades of industry expertise. The "free lunch" mentality severely hinders technological progress. An effective method to incentivize experts to voluntarily offer high-quality FBDs for paid use is the black box + interaction platform model. The black box protects experts' intellectual property, while its transplant-free application resolves how the module is utilized. The interaction platform addresses the motivation for experts to contribute their FBDs. A well-protected and widely adopted expert knowledge module will far outperform fragmented, isolated efforts by individual experts. A knowledge interaction platform is urgently needed but has yet to be realized.

3.1.1.2. Minimalist application of FBDs—transplant-free

How can highly encrypted black-box FBDs be applied in a transplant-free, minimalist manner? This is a challenge FlashControl must address. Unlike conventional FBDs, the application of black-box FBDs is facilitated through a standardized XML-format file called the Functional Module Management Shell. Key contents of this file are generated by the compiler, and the XML file comprehensively specifies the information required for the FBD application, as illustrated below in **Figure 4**:

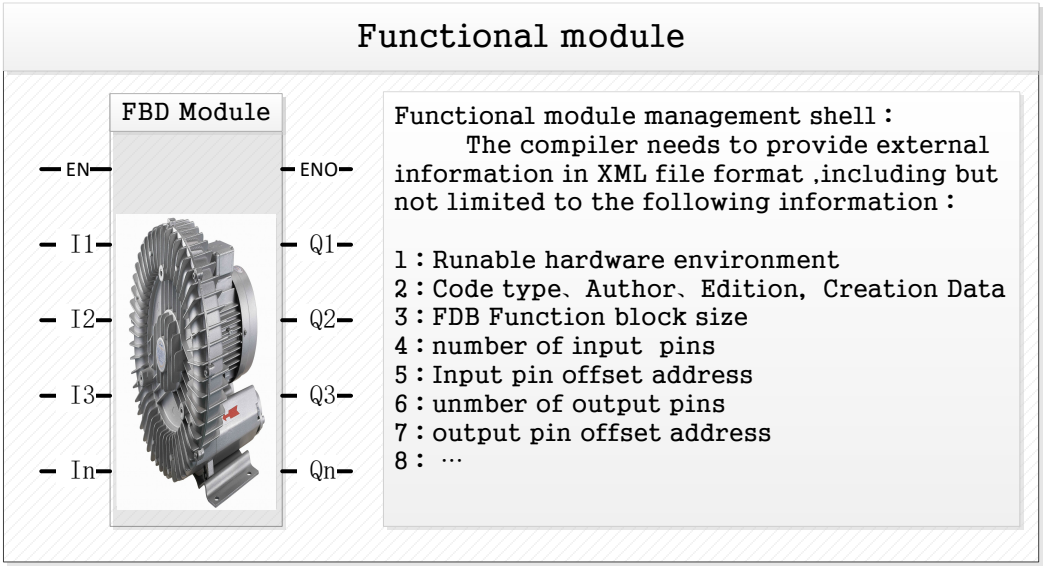

Figure 4. Schematic diagram of functional module management shell

Based on the information described in the XML file, FlashControl generates the Interface Data Sheet for this FBD (Function Block Diagram). Unlike conventional FBD execution methods, the FBD is positioned in memory by FlashControl's Organizer. Consequently, the runtime starting address of the FBD code block is also written into the Interface Data Sheet.

The Interface Data Sheet serves as the operational interface for the FBD, containing runtime-required data, attributes, pointers, flags, and storage units [3]. Its execution relies on matching methods, and the collection of these methods is termed: Parser.

The Interface Data Sheet constitutes only part of FlashControl's control data tables. The complete collection of control data tables is referred to as the Control Database. Thus, FlashControl operates as a data-driven system.

Figure 5 shows an example of the FBD interface data sheet is shown below:

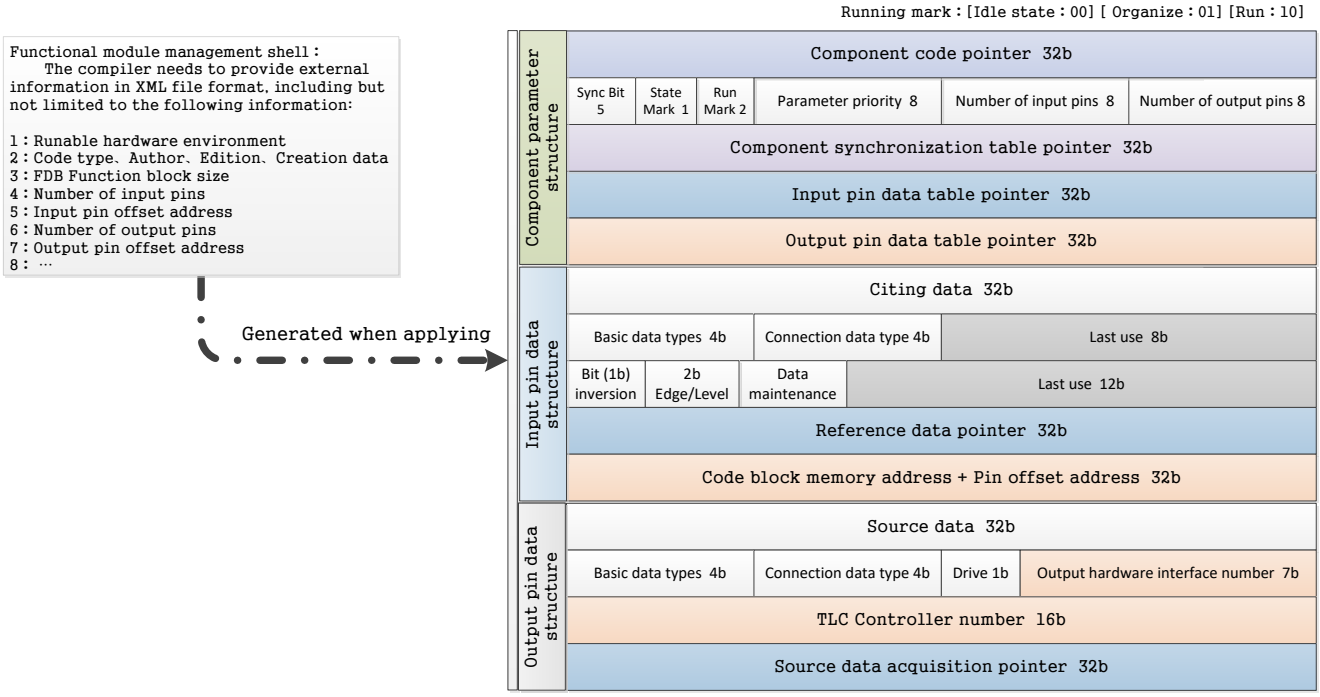

Figure 5. FBD interface data shows intention

3.1.2. Connection layer

FlashControl defines multiple controls for connecting FBDs, which are referred to as process controls. These process controls function similarly to logical statements in programming languages, serving as graphical representations for describing process logic.

The system-defined controls include: Start Control, End Control, Branch Control, Merge Control, Parallel Control, Process Call, and Component Control. These controls are represented by data tables in a predefined format. For example, the data structure definition of the Start Control is shown in **Figure 6**.

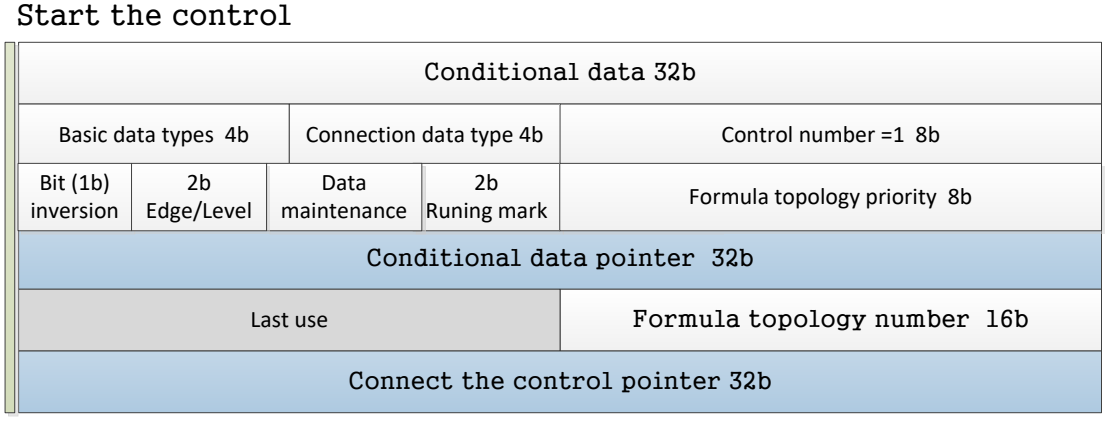

Figure 6. Data structure diagram of the start control

The data tables of process controls are also integral components of the control database. Controls are interconnected via linked list pointers, and their memory allocation is managed by the Organizer in FlashControl, as detailed in subsequent sections.

3.2. Hardware (resource) topology—basis for data movement paths

Hardware topology **refers to the network connectivity relationships among hardware modules within a** user's engineering scope. It is one of the core components of FlashControl. The intricate interplay of hardware topology is a root cause of complexity in process control, requiring control engineers to expend significant effort in programming to establish data reference relationships between hardware modules.

Unlike conventional interpretations of hardware topology, FlashControl observes data references within the process topology **and** leverages the network connectivity of hardware modules. The Organizer **in** FlashControl then automatically generates a series of standardized control data tables. These tables can be parsed by interpreters **embedded within hardware modules, eliminating time-consuming programming and enabling streamlined applications** [2]. FlashControl records the movement paths of data reference relationships from the process topology in the system's connection database.

The Example hardware topology is shown below as **Figure 7**.

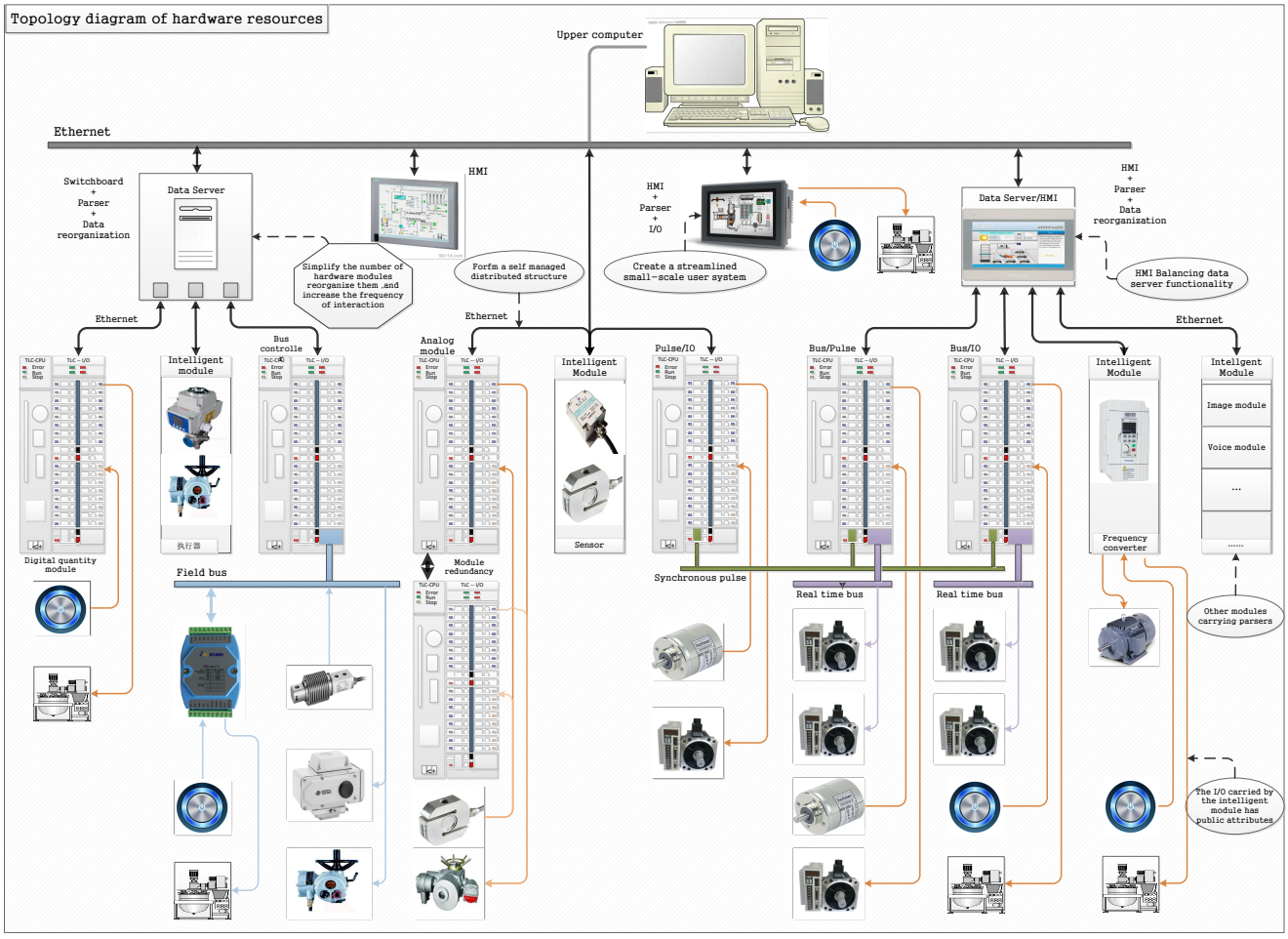

Figure 7. Hardware topology diagram

3.2.1. Network topology architecture

Hardware modules connected within the network topology can be linked through switches to form a flattened network structure. To accommodate existing fieldbus systems, a hardware module may also act as a fieldbus module. Fieldbus devices can connect to the user control system through such modules, creating a two-layer network topology architecture.

3.2.2. Open application

Open application of hardware modules **is a key trend in industrial control development** [4]. Integrating sensors, actuators, I/O modules, and other devices from diverse manufacturers into a unified user system significantly enhances control quality while reducing dependence on specific hardware vendors.

Imagine a blank canvas that allows free creation of poetry or art—this represents ultimate flexibility and openness. Can hardware module applications achieve such a "blank canvas" state? This is precisely the goal of FlashControl. A hardware module consists of two parts: the **hardware entity and the resource management shell**. While the hardware entity remains structurally conventional, the unique **resource management shell distinguishes it from typical hardware modules. The resource management shell is the sole means for** FlashControl to recognize, understand, and utilize hardware modules. On one hand, it describes the hardware's application and initialization information; on the other, it specifies memory and CPU resources where **control data can be allocated.**

Similar to software functional modules, the hardware resource management shell uses an XML file format to define the hardware module's application specifications. This file not only enables the generation of control data tables **required for engineering control but also ensures rational allocation and placement of these tables based on memory information.**

Generally, hardware modules from different vendors need only three elements to integrate into the system:

(1) An initialized Ethernet interface,

(2) A fixed data download module,

(3) A resource management shell file.

This setup closely resembles a "blank canvas," allowing the system to efficiently implement user-specific process control through a data-driven model.

The advantages of the data-driven model are evident:

(1) No compilation or linking is required.

(2) A bundled control data generation/allocation mechanism eliminates the need to write control programs for each hardware module in the user system.

(3) Simplified application and optimal control efficiency are achieved.

(4) Data flexibility strongly supports open automation [2].

The definition of the hardware resource management shell is only one part of enabling open applications, working in tandem with other system components. The structure and definition of a hardware module are illustrated in **Figure 8**.

The hardware modules of FlashControl execute control data for technology logic; therefore, they are also referred to as Technology Logic Controllers (TLCs).

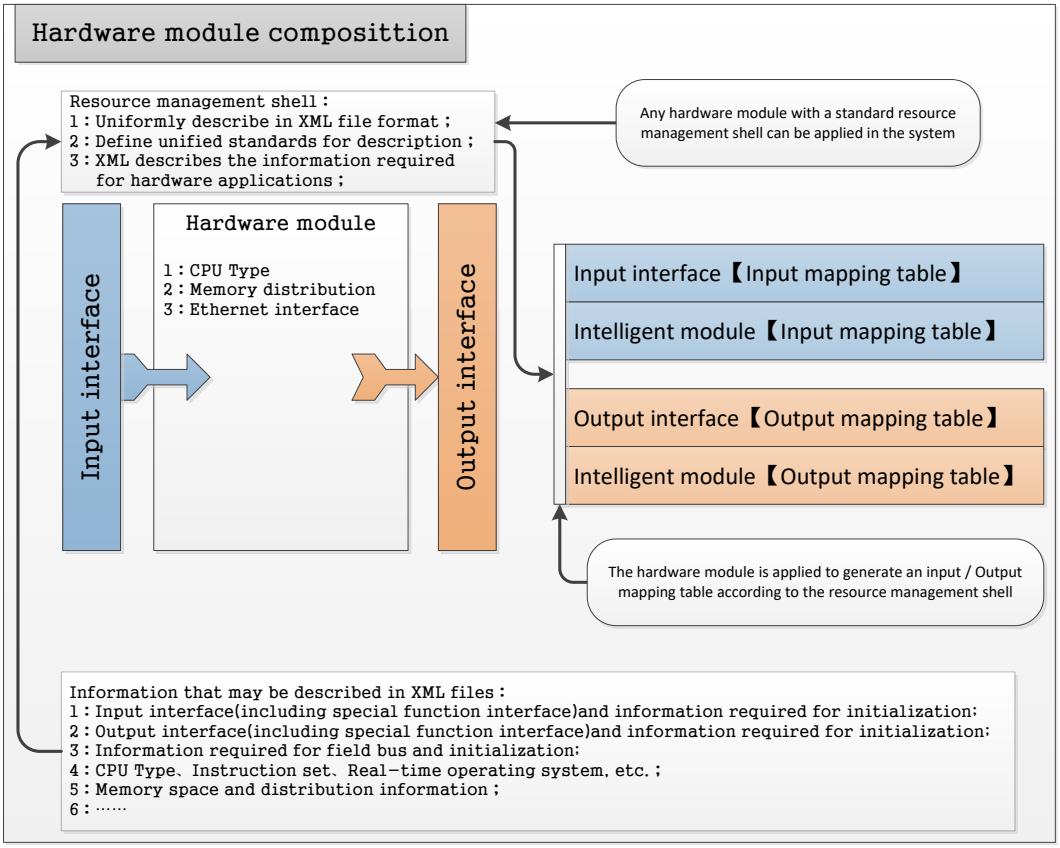

Figure 8. Hardware module application information diagram

4. Organizer—resource allocation & addressing

While connecting to the database records all information required for user process control, this information remains scattered and fragmented due to variations in user-specific processes, making it unintelligible to the CPU directly. The Organizer standardizes this control data into a predefined format compatible with the Parser for proper interpretation. The standardized dataset is termed the Control Database, which is executed by the Parser within the TLC hardware module. FlashControl's Organizer generates a dedicated Control Database for each TLC hardware module, structured as follows:

4.1. Project data header

During power-up, the system loads the Project Information Header into a predefined memory unit, as specified in the XML file of the Hardware Resource Management Shell. This header contains metadata about the user project, with a critical field being the location table base address. Since the Parser cannot inherently determine the memory locations of control data tables, which vary across user projects, the system creates a series of tables to store the base addresses of these control data tables. The Location Table Base Address is stored at a system-defined offset.

4.2. Location table

The location table facilitates addressing of control data tables. Organized in a predefined sequence, it enables the Parser to dynamically locate the base address of any required control data table during runtime.

4.3. Control data tables

Control data tables contain process logic interpretable by the Parser. These tables either supply runtime data or direct the execution flow of the Parser. While adhering to a standardized format, their content is process-dependent and varies with user-specific configurations. Consequently, the Parser needs only to execute fixed methods to derive the desired control outcomes. The schematic of the control database structure is shown below as **Figure 9**.

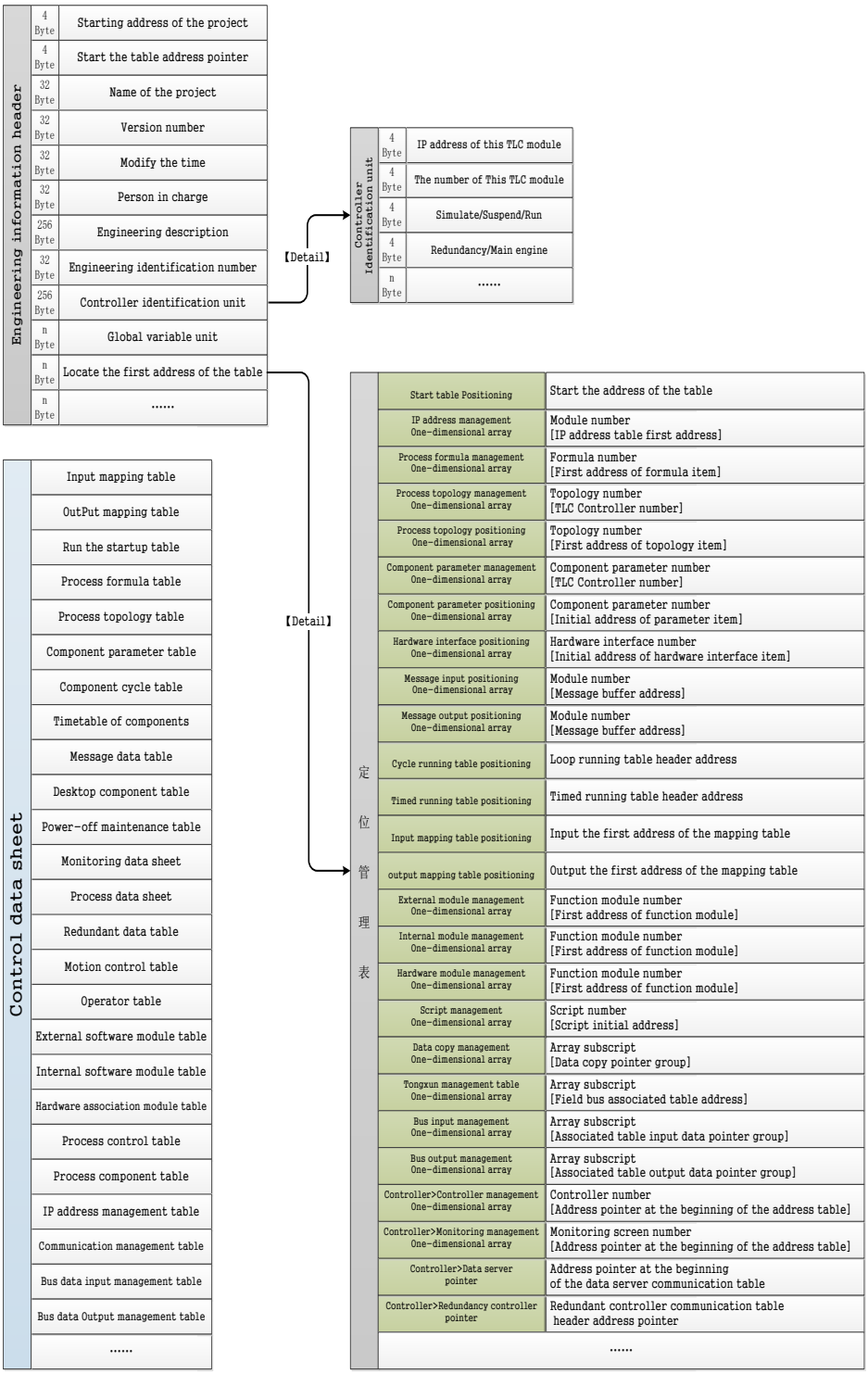

Figure 9. Control the database composition diagram

5. Parser—execution of control data

The Parser is a method repository for implementing user-specific process control through the Control Database (tables).

To draw an analogy, the Parser operates similarly to a multiplication table (e.g., the "9×9 table"). During application, it requires only personalized input data. By applying the predefined algorithms akin to the multiplication table's rules, it derives the required control outcomes. While input data may vary infinitely, the underlying algorithm remains singular. This encapsulates FlashControl's core philosophy, which fundamentally differs from conventional control paradigms. Rather than tailoring implementations to specific user processes, the Parser focuses on enumerable, universal methods required to achieve process objectives. These methods are finite and predefined. Consequently, the Parser itself is an immutable program, with execution details fully defined by the contextual data provided in the Control Database (tables) [3].

This design grants the Parser universal applicability in process logic execution. Pre-embedded in hardware modules, instruments, or devices equipped with the Parser require no additional programming. Once integrated into a user's system, FlashControl injects control algorithms directly. This paradigm delivers transformative value:

Minimalist implementation: Drastically reduces development time and resource expenditure.

Autonomous local control: TLC hardware modules autonomously collect data and execute localized control, embodying the principles of FCS (Fieldbus Control System) architecture.

Open control framework: Represents the future direction of instrumentation development, enabling seamless scalability and interoperability. The example of Partial Methods & Corresponding Control Data Table shown below, **Figure 10**.

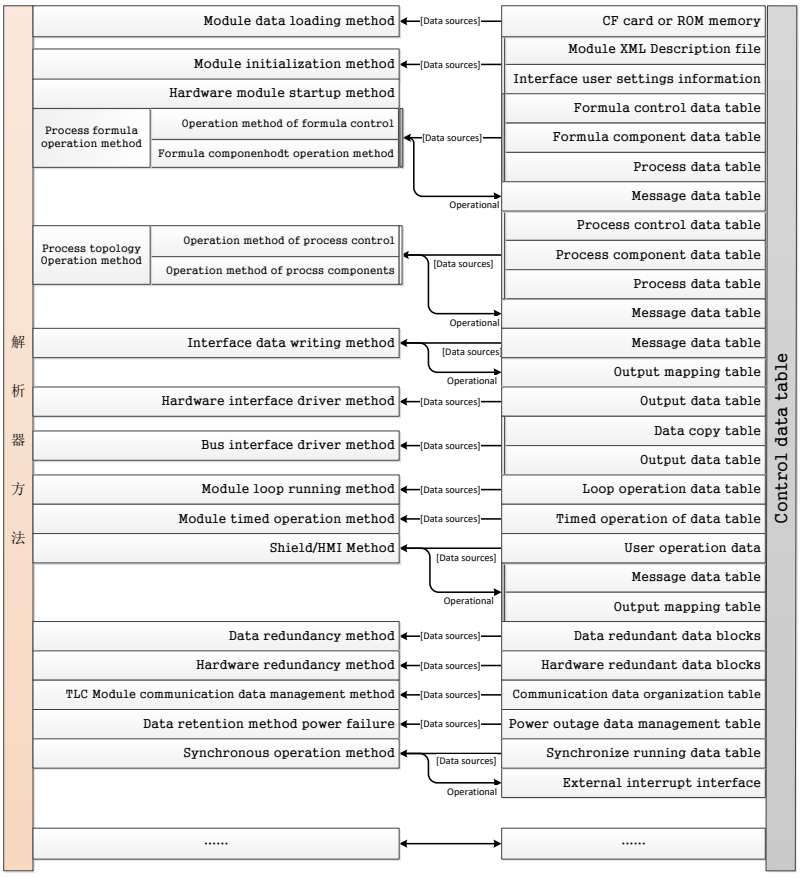

Figure 10. The method corresponds to the data representation intention

6. Flexible control

According to **Figures 11** and **12**, Flexible control is a challenging demand imposed by users on control systems, necessitating a flexible control mode capable of altering process control logic through the reconfiguration of upper-level control and management software such as MES (Manufacturing Execution System) and APS (Advanced Planning and Scheduling), thereby achieving flexible manufacturing objectives. A user control system can be composed of numerous sufficiently granular process topologies, each assigned a unique identifier. These identifiers can be logically reorganized within a recipe array to dynamically alter the process control logic. This approach simultaneously addresses the customization requirements of upper-level management systems like MES and APS, enabling dynamic adaptation of production processes while maintaining system modularity and reconfigurability.

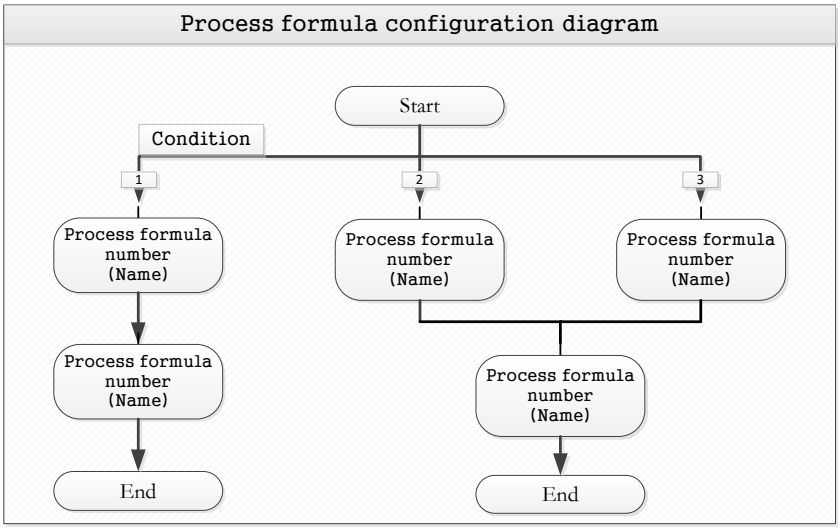

Figure 11. Process formula diagram

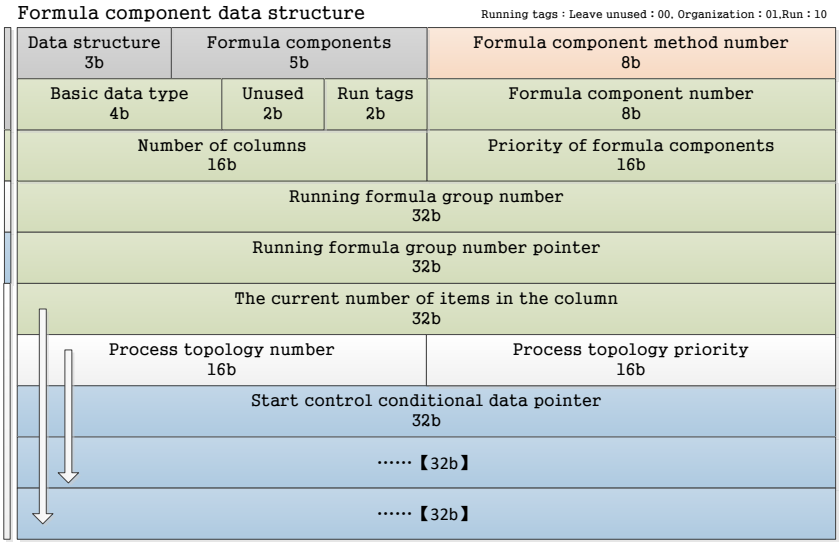

Figure 12. Formula component data structure

7. Motion control rethinking in data-driven mode

From **Figure 13,** the implementation of motion control in FlashControl does not rely on high-speed data

communication. Instead, it operates through the coordination of a motion control data table and the "wired-AND" relationship of the CPU's external hardware interrupts. The motion control data table is a standardized data structure that includes conditions triggering a motion and post-execution driving actions, enabling seamless integration with process control. Theoretically, n motion axes can work synchronously under this architecture, achieving precise and efficient multi-axis coordination without dependency on real-time communication protocols.

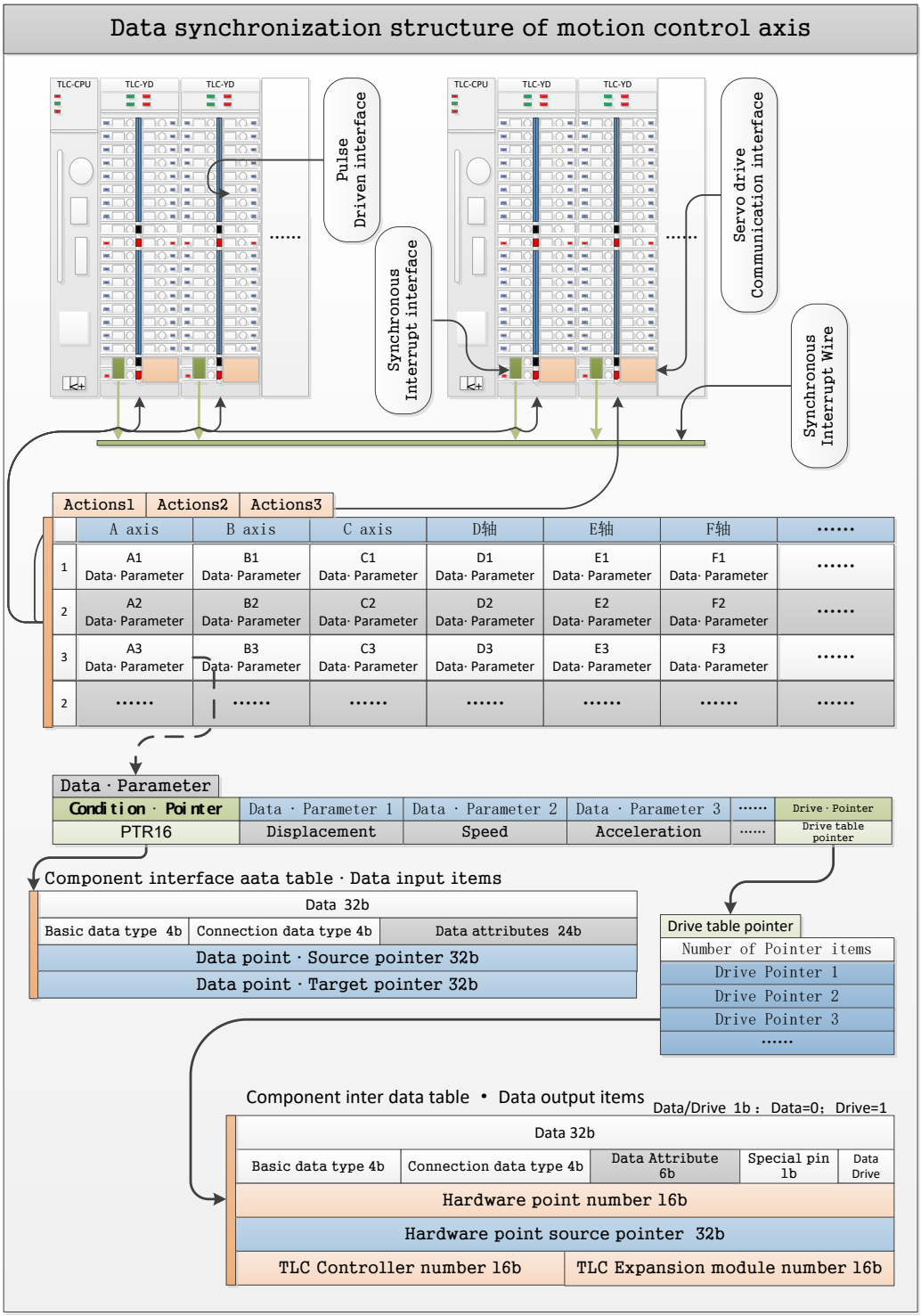

Figure 13. Schematic diagram of motion control synchronization

8. Conclusion

FlashControl re-examines the essence of control systems from the perspective of process engineers, adopting a user-centric engineering vision to propose an observer- and data-driven control paradigm, offering an innovative solution for control implementation. This approach not only simplifies and accelerates control realization but also achieves the following industry-critical objectives:

Open control architecture: Refers to the universality of software and hardware modules. After decades of fragmented control system development, achieving interoperability has become highly challenging. FlashControl addresses this by adopting XML-based descriptive configurations for its software and hardware modules, laying the foundation for open, modular applications.

Flexible control: Small-batch and customized production has become a focal industry topic. FlashControl enables dynamic process reconfiguration through its Process Topology framework, managed by a Process Recipe component. This allows upper-layer software systems (e.g., MES, APS) to dynamically reorganize processes via topology numbering, supporting agile manufacturing.

Rapid adaptability: Traditional fixed-program systems struggle to accommodate frequent process optimization, modification, or upgrade requests. FlashControl's data-driven mode enables dynamic and rapid adaptation to evolving requirements.

Enhanced maintainability: Complex control implementations exacerbate maintenance challenges caused by personnel turnover. FlashControl's simplicity and open architecture significantly improve system maintainability, aligning with end-user needs.

Process interaction-free design: Both process and control engineers can intuitively understand and directly implement control logic through Process Topology, eliminating time-consuming requirement negotiations between disciplines.

Process confidentiality: Unlike reliance on third-party expertise, where confidentiality depends on external integrity, FlashControl's simplicity enables self-sufficient implementation, ensuring highly reliable process secrecy.

Decentralized control architecture: Traditional distributed controllers require complex programming for inter-controller coordination. FlashControl's global control data allocation mechanism revolutionizes this paradigm, enabling practical implementation of decentralized control architectures without intricate coupling logic.

As a systematic and creative engineering breakthrough, FlashControl holds significant potential to advance industry technology. Its proof-of-concept demo has validated the feasibility of this control paradigm, yet further research is required to determine whether it can evolve into a groundbreaking control system framework. This vision awaits critical evaluation and exploration by industry professionals.

Disclosure statement

The authors declare no conflict of interest.

References

[1] Tang Z, 2014, *Industrial Process Control Rapid Generation System and Implementation Method*, China ZL201410423160.7, amended August 26, 2014, China, viewed April 15, 2025.

[2] Tang Z*, 2019, Method for Direct Control Implementation via Process Topology Diagram*, China 201910081956.1,

amended January 28, 2019, China, viewed April 15, 2025.

[3] Tang Z, 2017, *Process-Guided System for Industrial Process Control*, China ZL201710257632.X, amended April 19, 2017, China, viewed April 15, 2025.

[4] Peng Y, 2020, *Fully Open Industrial Automation Systems Are Emerging*, literature, *Knowledge Automation*.