electronics

Article

A Software-Defined Gateway Architecture with Graphical Protocol Modeling for Industrial Control Systems

Rong Zheng [1,2], Song Zheng [1,3,*], Chaoru Liu [2], Liang Yue [1] and Hongyu Wu [2]

[1] College of Electrical Engineering and Automation, Fuzhou University, Fuzhou 350108, China
[2] IAP (Fujian) Technology Co., Ltd., Fuzhou 350108, China
[3] Key Laboratory of Industrial Automation Control Technology and Information Processing, Fuzhou University, Fuzhou 350108, China
* Correspondence: s.zheng@fzu.edu.cn

Abstract

Within the context of Industry 4.0, the integration of heterogeneous industrial devices into unified control and supervision systems remains a fundamental challenge due to diversified communication protocols and interfaces. Conventional industrial gateways relying on customized driver development encounter issues such as high protocol extension costs, lengthy development cycles, and limited compatibility, restricting the agility and scalability of modern industrial embedded control systems. This paper proposes a novel paradigm for industrial interoperability gateways based on a software-defined architecture and graphical modeling. Through layered decoupling of software functions, protocol parsing and data conversion functionalities are encapsulated into draggable graphical components, enabling flexible adaptation and efficient debugging of heterogeneous protocols. The gateway middleware GMGbox architecture was designed and validated through an experimental platform. Results demonstrate that the gateway can accurately and concurrently parse multiple heterogeneous protocols via graphical configuration, stably acquire over 3000 data points, support online visual debugging and flexible deployment of protocol logic, and seamlessly integrate final data with upper-layer systems through standard protocols. Case studies of a level control system and a wastewater treatment plant Supervisory Control and Data Acquisition (SCADA) further validate the effectiveness and practical utility of this paradigm in real-world industrial scenarios. The proposed solution provides a novel architectural paradigm for building reconfigurable and maintainable systems in the field of industrial automation.

Keywords: protocol parsing; data conversion; graphical modeling; industrial gateway; interoperability; Industry 4.0

check for updates

Academic Editor: Antoni Morell

Received: 18 October 2025
Revised: 3 November 2025
Accepted: 5 November 2025
Published: 8 November 2025

Citation: Zheng, R.; Zheng, S.; Liu, C.; Yue, L.; Wu, H. A Software-Defined Gateway Architecture with Graphical Protocol Modeling for Industrial Control Systems. *Electronics* **2025**, *14*, 4369. https://doi.org/10.3390/electronics14224369

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The Industrial Internet is regarded as the core driving pathway of the Fourth Industrial Revolution. Its essence lies in leveraging networking and data-driven mechanisms to achieve deep integration of factory-floor devices, information management systems, and automation control systems, thereby realizing comprehensive connectivity among humans, machines, and physical objects. This developmental trend is profoundly reshaping the long-established, closed, and hierarchical architecture of industrial automation systems, significantly enhancing the overall operational efficiency of industrial control systems [1]. Currently, the Industrial Internet has entered a phase of rapid development [2–4], which is

reflected in its significant economic impact. For instance, by 2024, the number of connected devices in China's Industrial Internet had exceeded 100 million, while the value added by its core industries reached 5.01 trillion Renminbi (RMB) [5,6].

The paradigm of Flexible Manufacturing Systems (FMS) demands that industrial control systems be inherently agile and reconfigurable. However, this vision is severely hampered at the foundational level by the rigidity of traditional data acquisition infrastructure for industrial electronic devices (e.g., Programmable Logic Controllers (PLCs), drives, sensors). The high cost and long cycles associated with integrating heterogeneous devices through customization create a significant bottleneck, making the entire control system resistant to change. Therefore, achieving system-level flexibility necessitates a flexible data integration layer, which is the core focus of this paper.

However, the large-scale deployment of the Industrial Internet still faces significant challenges. A critical issue is the vast number of field devices in factories, accompanied by a wide variety of heterogeneous network protocols and communication interfaces. Total number of communication protocols used across different industrial sectors reaches into the thousands, including hundreds of widely used field-level protocols such as Modbus communication protocol, Transmission Control Protocol/Internet Protocol (Modbus TCP/IP), Process Field Bus (PROFIBUS) communication protocol, Ethernet for Control Automation Technology (EtherCAT), HostLink communication protocol, Factory Interface Network Service (FINS), Recommended Standard 232 (RS232), the Siemens S7 communication protocol (S7), EtherNet Industrial Protocol (EtherNet/IP), and Process Field Net protocol (PROFINET). Furthermore, communication protocols used by a large number of legacy devices are often variants of common industrial protocols, with their primary differences manifesting in the application-layer data format. This coexistence of multiple protocols severely restricts the interoperability capabilities among industrial devices [7,8]. The data silos formed by protocol fragmentation directly impedes the implementation of high-level control strategies, complicates system-wide monitoring and diagnostics, and increases the life-cycle cost of industrial electronic and automation systems.

Existing solutions primarily follow two technical routes: one adopts universal standardized protocols like Open Platform Communications Unified Architecture (OPC UA) and Message Queuing Telemetry Transport (MQTT) [9–14], while the other relies on driver-software-based middleware for protocol conversion [7,15–18]. Although universal standard protocols offer ideal unification, the cost of retrofitting legacy devices lacking native support is high, and the coexistence of multiple standards can introduce new integration complexities [14]. On the other hand, driver-based middleware solutions, even those using configurable text files based on the software-defined approach (e.g., eXtensible Markup Language (XML)), still suffer from limitations such as cumbersome configuration processes, difficulties in debugging, and insufficient scalability [15]. In recent years, graphical flow-based programming tools have emerged [19–24], providing new ideas for reducing development complexity. However, their applications in protocol interoperability is mostly confined to lightweight Information Technology (IT)/ Internet of Things (IoT) scenarios (e.g., Node-RED [25]), and they lack sufficient validation regarding support for complex industrial protocols and industrial-grade reliability.

To address the aforementioned challenges, this paper proposes a novel paradigm for an industrial interoperability gateway based on software-defined principles and graphical modeling. Differing from traditional protocol conversion gateways, the core innovation of this research is a breakthrough at the architectural level:

- A layered and decoupled, software-defined architecture is proposed to structure the gateway software into layers and modules, providing a unified runtime environment for protocol parsing logics.

- Business logic, including protocol parsing and data conversion, is decoupled from fixed drivers and reconstructed into programmable applications that can be configured via graphical modeling.
- The gateway middleware system GMGbox, embodying this paradigm, was designed and implemented. Systematic performance tests and case studies validate its significant advantages in enabling seamless access for heterogeneous devices and enhancing system flexibility and scalability.

The remainder of this paper is organized as follows. Section 2 reviews the state of the art in related technologies. Section 3 details the proposed paradigm, system architecture and implementation framework. Section 4 presents the experimental validation and the case studies. Section 5 discusses the differences between the proposed method and the existing approach. Finally, Section 6 concludes the paper and discusses future work.

2. Related Work

Interoperability challenges in industrial systems are primarily addressed through two technical approaches: the adoption of universal standard protocols and the use of driver-based middleware. This section reviews recent advancements in these technologies and highlights their limitations within dynamic industrial environments.

2.1. Universal Standard Protocol Technology

The first technical approach aims to achieve interoperability through the adoption of universal standard protocols such as OPC UA and MQTT. These protocols provide standardized frameworks for data exchange, enabling seamless integration between devices and upper-level systems. As an International Electrotechnical Commission (IEC) 62541 standard, OPC UA offers a secure, platform-independent framework for information modeling and communication [26]. It supports rich semantic modeling and real-time data access, making it suitable for complex industrial automation systems [9,10]. Its integration with Time-Sensitive Networking (TSN) can significantly enhance the determinism of real-time communication for industrial equipment, meeting the stringent timing requirements of industrial applications [11]. Similarly, MQTT, as a lightweight publish/subscribe messaging protocol, is ideal for environments with constrained network bandwidth or computing resources. Its simplicity and low overhead have established it as a de facto standard for cloud-centric Industrial Internet of Things (IIoT) data acquisition [12,13].

Despite their theoretical advantages, these universal standards face significant challenges in practical deployment. Legacy devices (e.g., PLCs, sensors), which form a substantial part of existing industrial infrastructure, often lack native support for modern protocols like OPC UA or MQTT [1]. Retrofitting these devices is often prohibitively expensive and technically challenging. Furthermore, as each standard comes with its own ecosystem, the coexistence of multiple standards can introduce new integration complexities rather than simplification [14]. Consequently, while universal standards are ideal for new greenfield projects, they alone cannot fully resolve interoperability issues in brownfield environments where compatibility with a multitude of heterogeneous protocols is essential.

2.2. Middleware Technology Utilizing Driver Software

To bridge legacy equipment with modern IT systems, driver-based middleware technology has been widely adopted. This approach involves constructing an adaptation layer that converts proprietary protocol data into a format recognizable by upper-layer applications. Early solutions typically employed dedicated drivers for specific protocol conversions (e.g., Profibus Decentralized Periphery (Profibus-DP) to Modbus) [27]. While efficient for predefined scenarios, these hard-coded drivers lack flexibility. They require sub-

stantial development effort and struggle to adapt to new or evolving protocols. To improve adaptability, subsequent research shifted towards configurable driver architectures based on text-based protocol description languages like XML and JavaScript Object Notation (JSON) [7,15–18]. For instance, Jiang et al. [7] introduced a software-defined gateway that decouples hardware, base software, and the application layer, using XML configurations to manage multiple protocol drivers. Liu et al. [15] proposed a service-oriented Industrial IoT gateway that utilizes text-based description languages such as XML and JSON to define data structures and interaction commands, enabling configurable drivers and flexible access for heterogeneous manufacturing equipment. Commercial products like the AVEVA PI Collector [16], Siemens Insights Hub [17], and KEPServerEX [18] also support numerous industrial protocols through configurable drivers.

However, these driver-based methods still face several limitations: high development complexity, as writing and validating protocol descriptions (e.g., in XML) demands specialized expertise and is time-consuming [15]; limited scalability, where even minor variations in existing protocols often necessitate writing descriptions from scratch, lacking a mechanism for logic reuse; and poor visibility and debuggability, since the protocol parsing logic is encapsulated within drivers or configuration files, making real-time monitoring, visualization, and debugging difficult.

2.3. Emergence of Graphical Tools

In recent years, graphical modeling and flow-based programming have emerged as new pathways to reduce coding complexity [19–24]. Among these, standards such as IEC 61131-3 and Scientific Apparatus Makers Association (SAMA) have effectively lowered the programming barrier in the field of industrial control [23,24]. Tools like Node-RED demonstrate the potential of visual programming in IoT integration [25,28,29]. The graphical notation for data acquisition architectures proposed by Trunzer et al. [30] further advances this trend. Nevertheless, most of these methods have been validated primarily in lightweight environments and lack support for complex industrial protocols; their applicability in high-reliability, real-time industrial scenarios remains to be proven. Although Node-RED has been applied to handle basic industrial protocols like Modbus TCP [28,29], its general flow-based programming model and component library lack native support for specific functionalities required by complex industrial protocols, such as device connection management, configurable-length cyclic polling, and structured parsing of complex data frames, posing persistent challenges in constructing industrially stable and usable critical control applications.

For a clear comparison of existing technologies, Table 1 summarizes the limitations of various technical paradigms for achieving interoperability among heterogeneous protocols in industrial equipment. As illustrated in Table 1, although both universal standards and driver-based middleware offer partial solutions, neither fully meets the demand for flexible, scalable, and low-cost interoperability in dynamic industrial environments. While software-defined approaches have found some application in the heterogeneous device integration and industrial service modeling [7,21], they have not been sufficiently applied at the level of protocol parsing and data conversion, nor have they been effectively combined with graphical modeling and modular architectures to enhance usability and reusability.

Consequently, this paper introduces a new gateway middleware paradigm that bridges the gap between the flexibility of graphical configuration and the rigorous reliability demands of industrial applications.

Table 1. Comparison of technical paradigms for industrial protocol interoperability.

Technical Paradigm	Primary Limitations
Universal standards	Relies on native device support; High retrofitting cost for legacy systems
Dedicated drivers	Hard-coded, lacks flexibility to adapt to new protocols
Configurable drivers (Text-based)	Complex configuration requires expertise; Difficult to debug
Existing graphical tools	Mostly target lightweight IT/IoT scenarios; Lack industrial-grade robustness
The proposed method	Aims to balance development efficiency, flexibility, and scalability, with support for legacy devices

3. Materials and Methods

3.1. Overall Design

This paper designs a novel industrial interoperability gateway middleware (GMGbox). Unlike traditional protocol conversion gateways, it applies the concepts of software-defined and graphical modeling to protocol parsing and data conversion in interoperability gateways. It uses a decoupled modular program design and graphical modeling components to flexibly extend the functions of device protocol parsing and data conversion. Additionally, it supports converting collected protocol data into standard protocol formats, meeting the requirements for Industrial 4.0 device integration.

3.1.1. Design Concept

The software-defined paradigm plays a critical role in the design of GMGbox. Centered on software, it employs layered system design and modular software decoupling methods to realize the key capabilities of interoperability gateways. As shown in Figure 1, the gateway adopts a layered architecture design. The resources of the underlying hardware and fundamental software (Linux kernel and device drivers) are abstracted and encapsulated, providing services to the upper layers via a set of standard runtime Application Programming Interfaces (APIs). Building upon these APIs, the gateway application software layer implements core business logic such as device protocol parsing, data storage, and data conversion. Within the application software layer, GMGbox adopts a decoupled modular approach to further divide the application software into fixed common modules like communication management, the graphical modeling program engine and the database, and user-configurable distinct parts.

The configurable parts of traditional software-defined gateways often achieve differentiation by configuring communication parameters and loading multiple pre-integrated independent protocol drivers [7]. The range of configurable functions is typically limited to selecting protocol driver modules and configuring communication addresses using XML. Unlike traditional methods, GMGbox improves the driver library for different protocols by further decoupling the protocol adaptation functions for heterogeneous devices from independent protocol drivers. These functions are implemented through a protocol dictionary library, a graphical protocol algorithm library, physical interface drivers such as User Datagram Protocol (UDP), TCP and serial ports. Since different device protocols can share physical interface drivers, they can be integrated into the foundational software layer, providing standard service interfaces for applications. The protocol dictionary library integrates dictionary templates for mainstream industrial device protocols used in industrial control fields, such as Modbus TCP/IP, EtherCAT, hostlink, and S7. These are stored on disk in file format for dynamic loading and unloading. Given that most device protocols in legacy industrial systems are derived from mainstream industrial protocols, their differences are reflected in data formats. Therefore, GMGbox further modularizes

and visualizes protocol parsing and conversion functions. This is achieved through flexible graphical modeling configuration tools and program engine modules, which implement protocol algorithms for specific protocol adaptations.

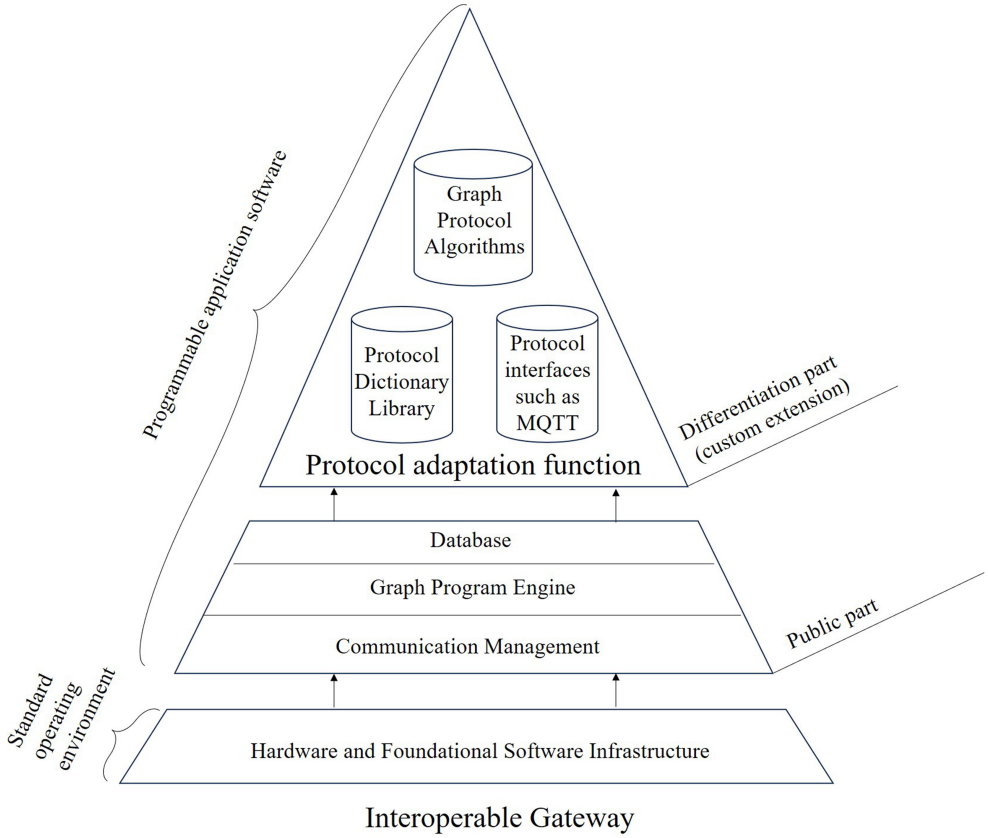


Figure 1. Conceptual architecture of the interoperability gateway based on software definition and graphical modeling.

3.1.2. Overall System Architecture

The overall framework of the system, as shown in Figure 2, is divided into three levels: hardware, foundational software, and application software. In this architecture, the hardware and foundational software such as the operating system only provide the necessary supporting services for the application software. The role and function of the gateway device are determined by the application software.

The hardware system uses Advanced RISC Machines (ARM) or Intel x86 architecture (X86) general-purpose hardware chips, multi-core Central Processing Units (CPUs), and solid-state drives. It supports various network and serial Input/Output (I/O) modules, including serial interfaces such as RS232 and Recommended Standard 485 (RS485), as well as Wi-Fi, 4G, and Registered Jack 45 (RJ45) for Ethernet, meeting the hardware resource requirements for communication collection and protocol conversion of various field devices.

The foundational software consists of a real-time embedded Linux operating system kernel and device physical interface drivers. It provides hardware-level isolation and a standard runtime environment for the application software. The Linux operating system software includes basic software modules such as file systems, process management, firmware drivers and memory management. The periodic data collection process also involves interaction with physical interface drivers. Device physical interface drivers provide various communication software interface functions. They can initialize interfaces, receive interface configuration parameters such as data bits and parity checks, and use protocol stacks like TCP/IP, UDP, or serial communication to send and receive data. They also

provide error handling mechanisms for exceptional situations and status query functions for interface connection/disconnection and data reception.

The application software adopts a modular design, consisting of a protocol dictionary library adaptable to various industrial protocols, a graphical modeling program engine, embedded databases and caches, standard protocol modules (e.g., OPC UA, MQTT, Hypertext Transfer Protocol (HTTP)), and a communication program management module. OPC Unified Architecture (OPC UA), serving as the standard communication protocol in the Industry 4.0 reference architecture RAMI 4.0, can be integrated to achieve seamless interoperability between heterogeneous industrial devices and external systems. Different functional modules can be invoked by defining parameters such as interface modules and protocol types, achieving differentiation in business functions like protocol parsing, dictionary management, and data conversion for the gateway middleware. This improves the system's flexibility and scalability in integrating heterogeneous devices.

To adapt to the complex and variable protocol communication environment in industrial fields, such as the expansion of industrial control systems, the diversification of data formats of newly connected industrial devices, and the sharp increase in communication points, the designed gateway system is equipped with graphical modeling configuration software and gateway management software that can run on supervisory machines. The mapping relationships between different bit parameters in industrial data acquisition transmission frames can be defined by graphical modeling software, described using a function block graphical modeling language. This software provides a visual parameter configuration environment, enabling the visual configuration and dynamic adjustment of parameter information. The transmission sequences of standard protocols (e.g., OPC UA, MQTT, HTTP) are defined by the gateway management software, supporting the configuration of protocol communication parameters using XML.

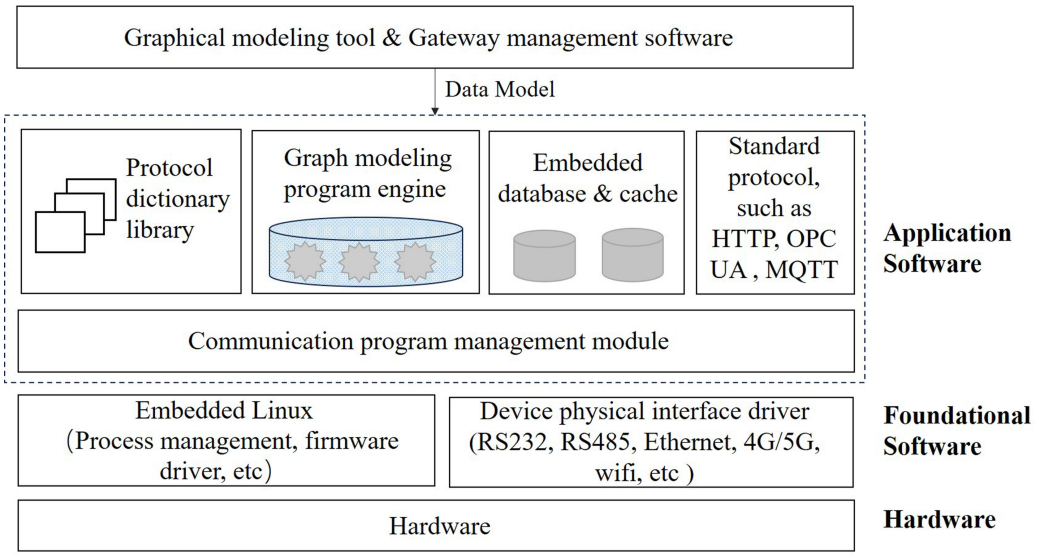


Figure 2. Overall system framework of GMGbox.

As shown in Figure 3, the communication program management module is the core part of the application software. It manages different types of protocol programs, such as system initialization and startup, creating and destroying device connections, and forwarding communication messages for device data collection and control command delivery. Additionally, it is responsible for monitoring each functional module in the system, managing and controlling the execution order between different modules and the data flow within the system according to preset scheduling methods. It coor-

dinates the functional modules to complete the entire process of protocol parsing and data conversion.

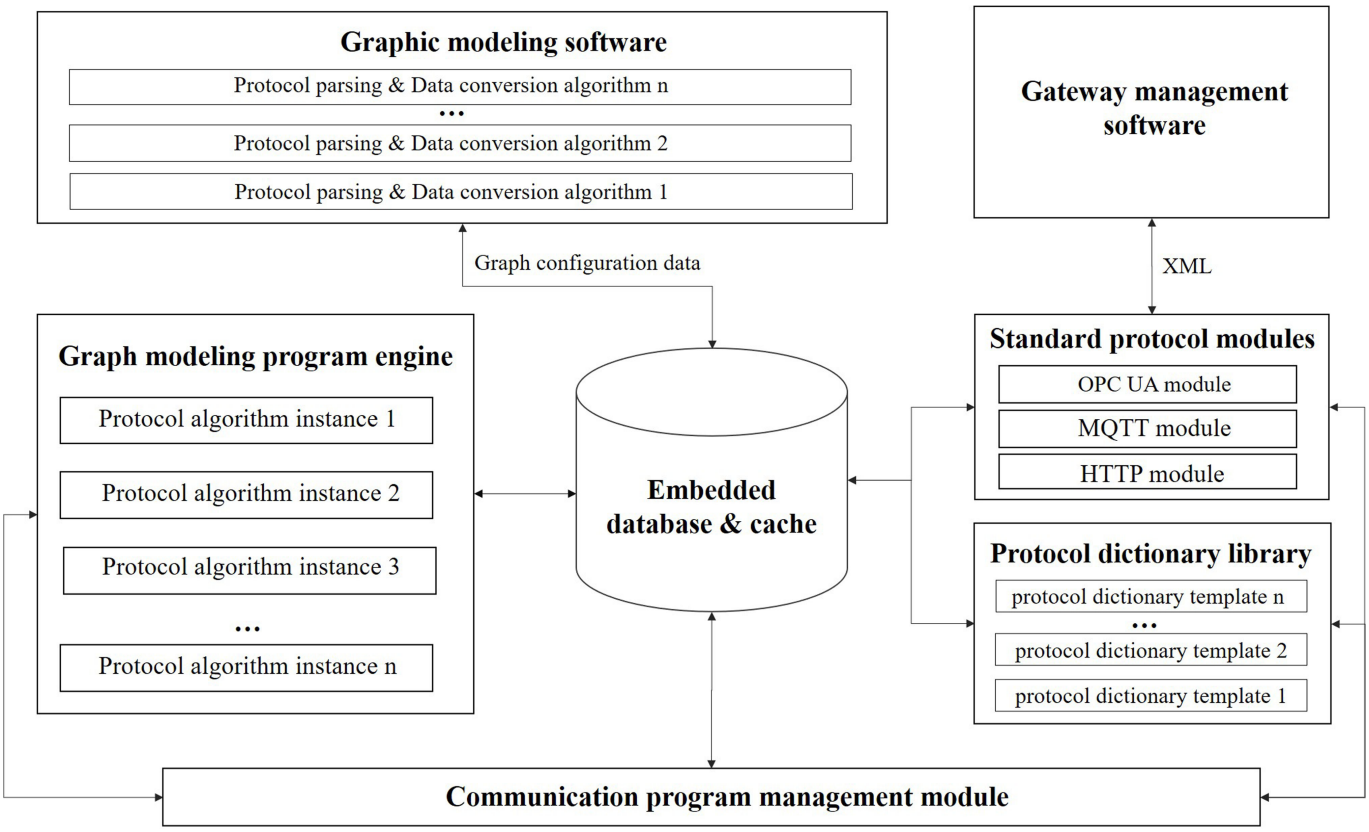


Figure 3. Core functional modules and their interactions in GMGbox.

The embedded database is another core component of the application software. This module serves as the data exchange center for various functional modules such as protocol algorithm program instances, the protocol dictionary library, and standard protocols services. It provides real-time/historical data, configuration data, and communication configuration data caching functions, and supports in-memory data storage. Additionally, it uses fixed data structures to describe and map common features involved in graphical modeling program models (e.g., component parameters, component addresses, calculation cycles, successor component addresses) and allocates and accesses them using fixed addresses.

Unlike traditional XML text descriptions, the communication logic and parameter configuration required for acquiring, parsing, and converting different types of industrial protocol data during gateway usage can be accomplished through graphical modeling components and drag-and-drop connections. This approach reduces the difficulty of customized development for parsing and converting heterogeneous industrial protocol data formats and quickly expands the business functions of the gateway middleware.

3.1.3. Gateway Workflow

Figure 4 shows the workflow of GMGbox interoperability gateway. The basic workflow of the gateway is as follows: First, the graph modeling program engine converts parameters (such as the target device address, packet length, etc.) into protocol request data based on the pre-configured graphical protocol algorithm logic, and places this data into a designated buffer. Subsequently, the communication program management module reads these request data, loads the protocol dictionary, generates device connection and communication task instructions. It then invokes the underlying physical driver interface to forward and manage

the communication data, thereby enabling the storage and transmission of the collected data. Next, the graph modeling program engine, in conjunction with the received response data and protocol algorithm instances, parses and converts the data portion of the transmission frames, storing the results in an embedded in-memory database. Simultaneously, the communication program management module loads OPC UA/MQTT/HTTP modules according to the pre-configured standard protocol XML configuration information to facilitate the transmission of uplink and downlink data.

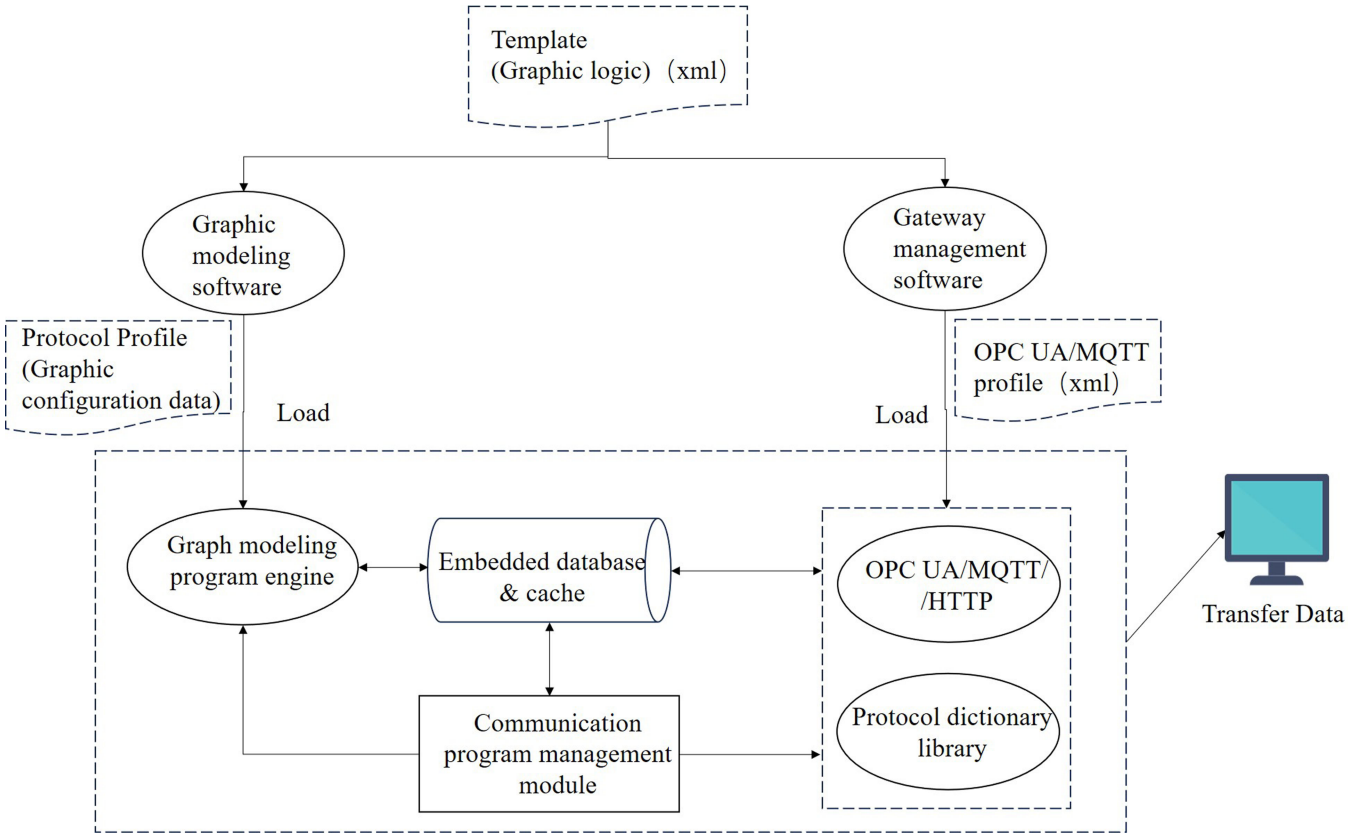


Figure 4. Main workflow of GMGbox.

In contrast to traditional XML-configurable gateways, the proposed interoperable gateway, by virtue of its modular design featuring hierarchical decoupling and graphical modeling. It provides a more flexible graphical modeling configuration tool and management module. This framework can generate configuration information, protocol parsing, and conversion algorithm instances tailored to different devices. It supports the batch reading of data packets, as well as the parsing, display, and conversion of data values from specified points. Furthermore, the gateway facilitates online modification and debugging of protocol algorithm instances and enables conversion operations with standard protocols such as OPC UA, MQTT and HTTP. Consequently, this solution is highly applicable to a wide range of Industry 4.0 scenarios, particularly meeting the demands of large-scale device data acquisition.

3.1.4. Protocol Parsing Based on Graphical Modeling

Generally, the industrial protocol communication process includes several key steps, including establishing a communication connection with the specified device, sending a communication task request, obtaining the original data packet, and parsing and converting the initial values of the communication data elements in the data packet. As shown in Figure 5a, traditional protocol parsing methods require offline development of drivers for

different protocols one by one and integrating them into the gateway. The processes of communication interface, protocol parsing, data mapping, and conversion in the drivers all require code programming and external debugging, which is extremely disadvantageous for large-scale device protocol integration.

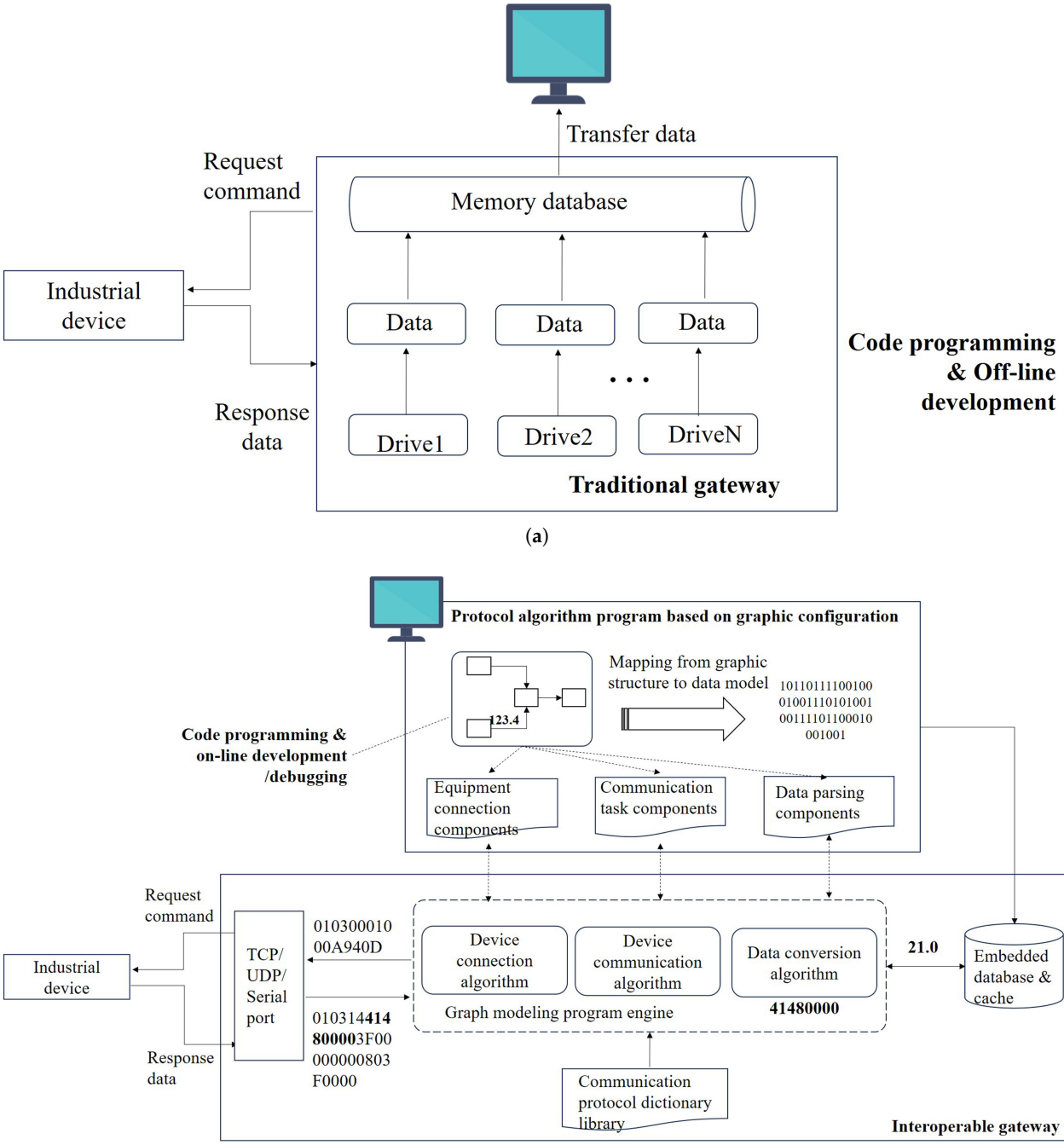


Figure 5. Comparison of protocol parsing and data conversion methods: (**a**) protocol parsing method based on driver code programming; (**b**) protocol analysis method based on graphic modeling.

Figure 5b shows the protocol parsing method based on graphical modeling programming. This method designs various graphical modeling components, such as device connections, communication tasks, and data parsing. It uses graphical modeling components to configure different protocol algorithm programs to achieve parsing and conversion of heterogeneous protocols. Given that the parsing of serial device protocols such as TCP, UDP, and RS232 requires addressing between devices through information such as IP addresses and communication cycles, GMGbox can first trigger a communication connection with the target device through device connection components and their configuration parameters, and provide real-time feedback on the device connection status. Subsequently, the communication task component generates protocol-specific instructions based on key parameters such as read/write operation commands, target address areas, data length, and protocol type for each distinct task. This process lays the foundation for requesting multiple data points in a single operation. Finally, the data parsing and conversion component can batch extract the raw values of multiple data units from a single response frame. These raw values are then converted into a set of target data values with specific meanings. The core mechanism enabling high-efficiency batch reading is this synergy between "configurable communication length" and "batch data extraction during parsing".

3.2. System Implementation Framework

3.2.1. Hardware and Basic Software

Selecting the foundational hardware for the gateway requires balancing performance and cost. The gateway hardware selected in this paper should support the deployment of the operating system and gateway business application software within controllable cost conditions, meet the usage requirements of most industrial field application environments, and also have good expansion interfaces. Considering that X86 processors have strong computing performance and mature software systems, they are widely used in industrial scenarios that require complex computing and stability. ARM processors, with low power consumption, miniaturization, and relatively low production costs, are especially suitable for industrial applications with high energy efficiency requirements and limited space. Therefore, this paper chooses X86 and ARM processors as the foundational hardware for the gateway. There are no strict restrictions on the selection of hardware devices for the gateway. As long as the relevant technical specifications are met, it supports various interfaces such as RS232, Recommended Standard 422 (RS422), RS485, and RJ45. The gateway uses a tailored Linux operating system and physical interface drivers for serial, TCP, and UDP protocols as its underlying foundational software. The hardware technical parameters of the gateway are detailed in Table 2.

Table 2. Gateway hardware technical parameters.

Parameter	ARM Gateway (UniMAT, Shenzhen, China)	X86 Gateway (EVOC, Shenzhen, China)
CPU architecture	ARMv7 sun8i (Cortex-A7) 1.2 GHz	4-core Intel Celeron J1900, 1.99 GHz
Internal memory	256M DDR3 RAM	4G DDR3 RAM
Hard disk	256M NAND FLASH	100G SSD
Network interface	RJ45, Wi-Fi/4G/5G	10/100/1000 Mbps RJ45
Serial interface	RS232, RS485	RS232, RS422, RS485

3.2.2. Application Software Design

In the design of the interoperability gateway, the protocol parsing and conversion functions for heterogeneous devices are accomplished through the flexible combination of application software modules, including the standard protocol dictionary library, the graphical modeling program engine, and a series of graphical components (the functions and design

features of each core module are listed in Table 3). This highly modular and functionally decoupled architecture is the core of realizing the software-defined gateway paradigm. As shown in Table 3, each module has a clear responsibility: the protocol dictionary library provides a unified abstraction of protocol details; the graphical modeling program engine is responsible for scheduling and executing algorithms; the communication management module coordinates internal data flow; the embedded database acts as a real-time data hub; and the standard protocol module enables standardized integration with external systems. Crucially, key communication processes such as device connection, communication task deployment, and data parsing are encapsulated into visually configurable graphical components. This allows users to complete functional configuration through intuitive drag-and-drop and connection actions, thereby replacing traditional code writing.

Table 3, which outlines the core software functions and design characteristics of the gateway, indicates that this architecture effectively separates business logic from underlying communication details through well-defined interfaces and clear functional boundaries. Based on this design, the proposed method possesses extensive adaptability to a variety of heterogeneous network protocols and physical interfaces. Although the communication protocols of different industrial devices exhibit significant differences in frame structure and semantic rules, their communication behaviors can be abstracted into a set of universal, configurable algorithmic processes (corresponding to the graphical components listed in Table 3). The differences mainly manifest in the attribute parameters of each component (such as protocol identifier (ID), device memory address, data encoding format, etc.), which constitute the core content for visual configuration within the graphical modeling environment.

Table 3. Gateway core software functions and design features.

Software Module	Main Function	Design Characteristics
Protocol dictionary	Provides standardized JSON templates describing protocol details	Unifies protocol abstraction, reduces extension difficulty, supports dynamic loading
Graphical modeling program engine	Loads and executes protocol algorithm models	Executes graphical parsing logic, supports dynamic model updates, implements cycle-driven parsing
Communication manager	Manages system initialization, device connections, data forwarding	Core communication hub, coordinates workflows, provides connection management
Embedded database & cache	Stores real-time data, configuration, communication settings	High-efficiency memory access, unified data storage interface
Standard protocol module	Converts data to standard formats (HTTP, OPC UA, MQTT)	Enables standardized integration with upper-layer systems
Device connection component	Configures connection parameters, manages physical connections	Visual connection configuration, decouples connection logic from parsing
Communication task component	Configures task parameters, generates protocol commands	Visual definition of data requests, supports batch reading configuration
Data parsing & conversion component	Configures parsing parameters, extracts and converts data values	Visual configuration of parsing rules, enables batch data extraction

The protocol configuration method based on graphical modeling constitutes a key advantage of this interoperability gateway. For the numerous protocol variants present in field devices, system integration engineers can rapidly construct device connection establishment, communication message parsing, and data conversion logic as well as perform online debugging and dynamic reconfiguration within the visual integrated environment, without relying on developers to write code. This paradigm shifts from "code programming" to "graphical configuration" significantly lowers the technical barrier and time cost associated with developing drivers for heterogeneous protocols.

To further enhance the reusability and reconfiguration efficiency of the protocol configuration programs, this study adopts a model-driven approach oriented towards graphical configuration. This method encodes the protocol algorithms, obtained through visual editing, into a configuration data model with a fixed structure and computational timing tags, which is dynamically loaded and executed by the graphical modeling program engine [31–33]. The encoded algorithm model is not only easy to store and reuse, reducing the complexity of engineering deployment, but more importantly, it introduces online debugging capabilities. Engineers can thereby abandon the cumbersome traditional process of "capturing data packets-manual parsing-coding drivers". Even when the communication packet structure is not fully known, they can quickly verify and optimize the parsing algorithm through real-time data monitoring and feedback, thereby significantly improving the flexibility of the debugging process and overall development efficiency.

4. Results

4.1. Experimental Validation and Case Studies

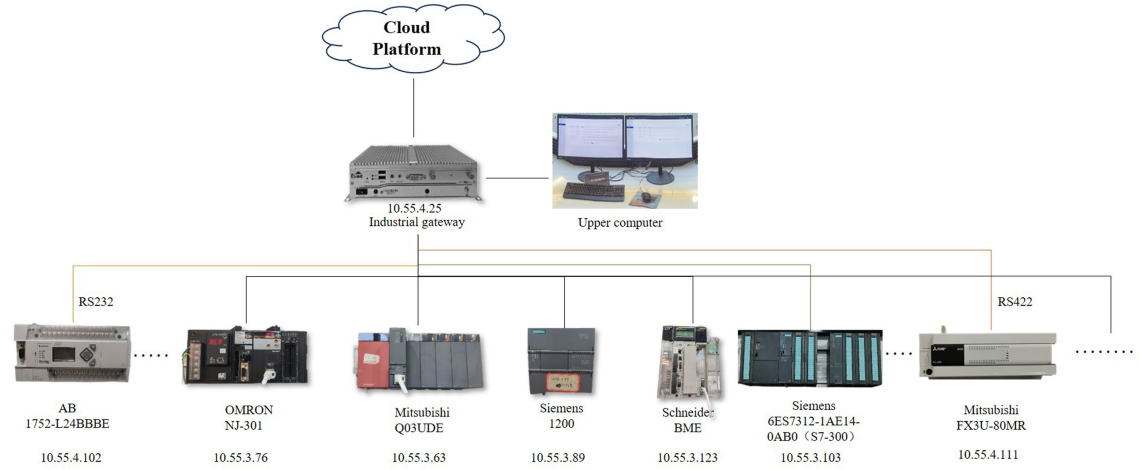
To evaluate the functionality and performance of the proposed software-defined interoperability gateway, an experimental test platform was constructed. This platform integrates PLCs from different manufacturers, industrial gateways, and a cloud server, enabling the deployment of heterogeneous protocol parsing programs, configuration of data acquisition parameters, and simulation data generation. Based on this platform, comprehensive functional and performance tests were conducted on the GMGbox gateway.

4.1.1. Experimental Environment

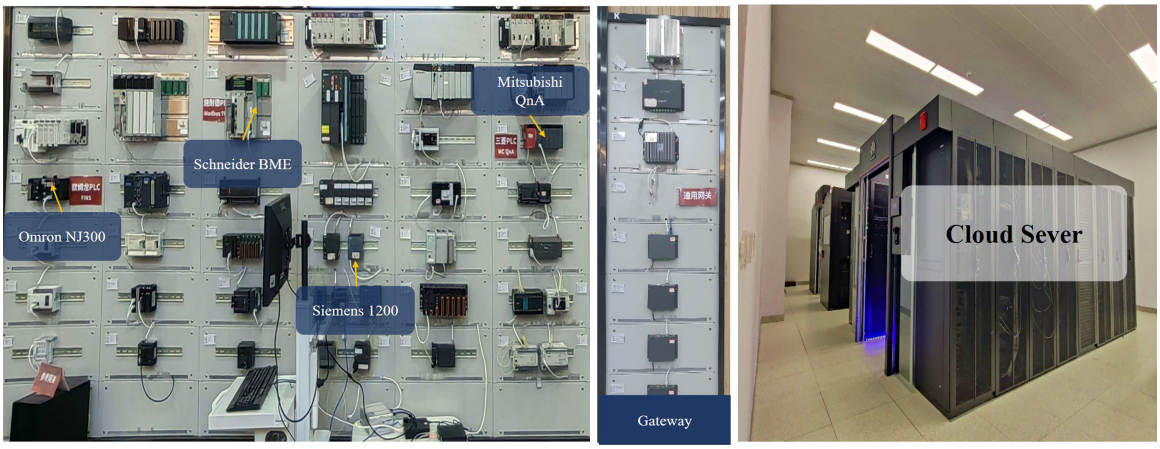
The gateway testbed designed for this study is illustrated in Figure 6. The hardware system of this testbed consists of various PLC controllers (including brands such as Omron, Mitsubishi, Siemens, and Schneider), industrial gateways (x86 and arm architectures), a cloud server, an upper computer (PC), network equipment, and other auxiliary devices, which are used for unit testing of the gateway software. The two gateway prototypes and their hardware parameters are detailed in Figure 6c and Table 2. In addition, four PLCs (Omron NJ300 (Omron, Kyoto, Japan), Mitsubishi Q03UDE (Mitsubishi Electric, Tokyo, Japan), Siemens S7-1200 (Siemens, Munich, Germany), Schneider BME (Schneider Electric, Rueil-Malmaison, France)), one x86 architecture gateway, and the cloud server were selected to form a test subsystem for verifying the gateway's protocol parsing and conversion capabilities (detailed hardware parameters of the PLCs are provided in Table 4). In this setup, the industrial PLCs are connected to the gateway via Ethernet. Communication between the gateway, PC, and cloud servers is established using the Internet. The PC runs the graphical modeling configuration software and gateway management tools. The industrial gateway is deployed with the gateway middleware software designed in this study. The cloud server hosts a set of industrial data simulation programs that automatically write data to the local four PLCs, simulating the data sources of field devices in an industrial setting.

Table 4. PLC controller types and protocols.

PLC Number	PLC Brand Type	Protocol
PLC A	OMRON NJ-301	Fins-UDP
PLC B	Mitsubishi Q03UDE	MELSEC-QNA
PLC C	Siemens 1200	S7-TCP
PLC D	Schneider BME	Modbus-TCP



(**a**)



(**b**)



(**c**)

Figure 6. Experimental system: (**a**) network architecture; (**b**) physical diagram; (**c**) gateway Prototype.

4.1.2. Gateway Test Process and Results

To evaluate the industrial applicability of the proposed gateway architecture, this study selected an Omron NJ-301 PLC (using the Fins-UDP protocol) as the target device. Identical graphical protocol parsing programs and protocol dictionaries were deployed on two gateway prototypes based on x86 and ARM architectures respectively, to acquire five real-time data points from this PLC while simultaneously monitoring the hardware resource utilization of the gateways. The experimental results (Figure 7) demonstrate that gateways with both architectures successfully parsed and monitored the PLC's real-time data correctly through graphical configuration. The graphical configuration process can be completed within 15 min, with the overall system resource consumption remaining at a low level. Even in the resource-constrained ARM-based device, the average CPU idle rates reached 81.44%, indicating the system's potential capability to handle larger data scales and more complex protocol tasks.

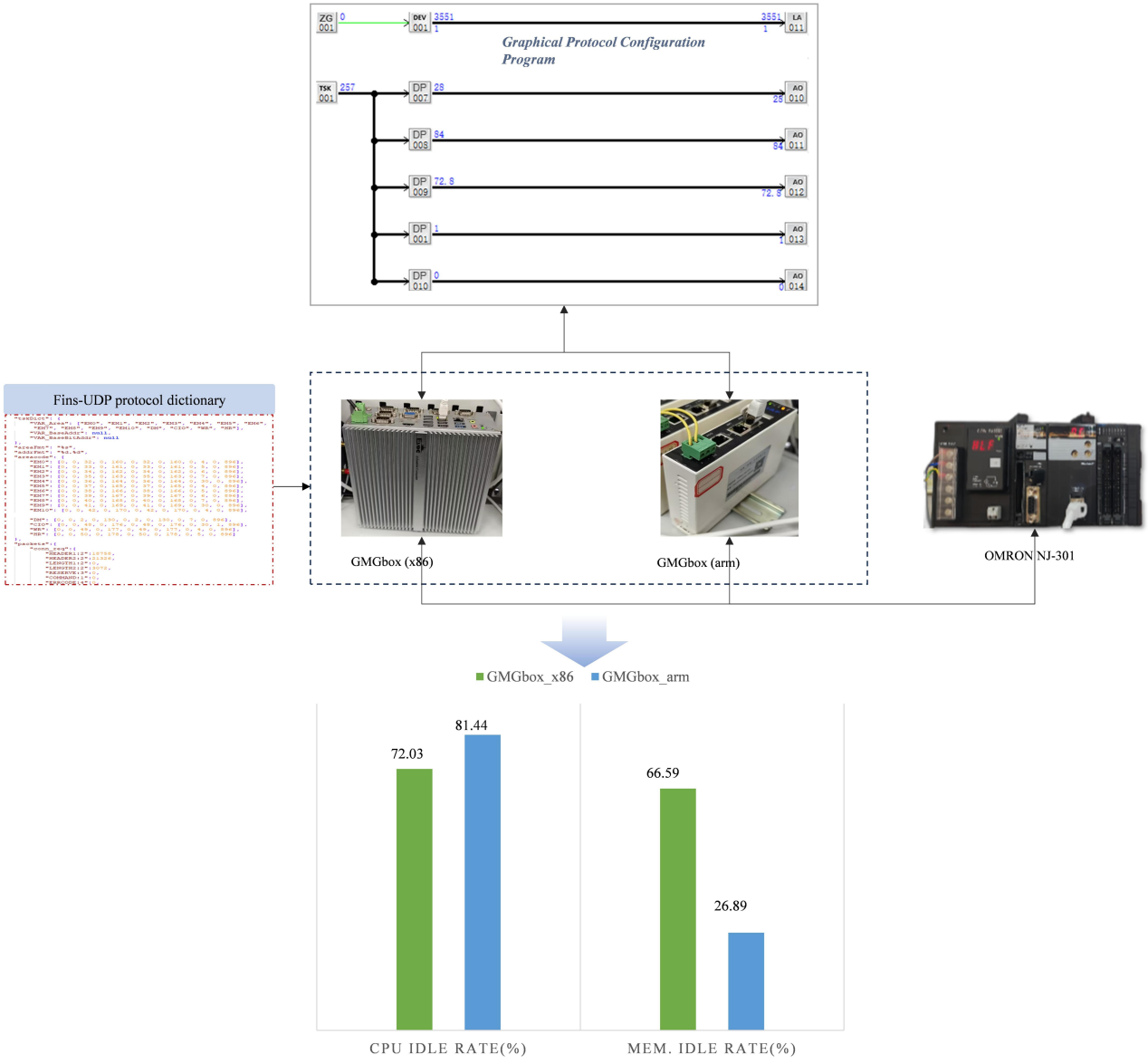


Figure 7. Basic function tests of the two gateways.

To validate the comprehensive performance of the proposed software-defined gateway in large-scale industrial access scenarios, a series of key capability tests were conducted.

These tests encompassed heterogeneous protocol data acquisition, data acquisition success rate, data point capacity, system stability, concurrent load handling, and network anomaly management capabilities.

For validating the heterogeneous protocol data acquisition capability, three PLC controllers supporting different protocols were selected as data acquisition targets: the Mitsubishi Q03UDE, Siemens S7-1200, and Schneider BME. During the testing process, the protocol parsing algorithm logic was first programmed using the graphical modeling and configuration software on the PC and then loaded into the gateway for execution. Subsequently, the same configuration software and a Web client were used to monitor the parsed and converted data values and their dynamic trends in real-time. The experimental results are shown in Figure 8.

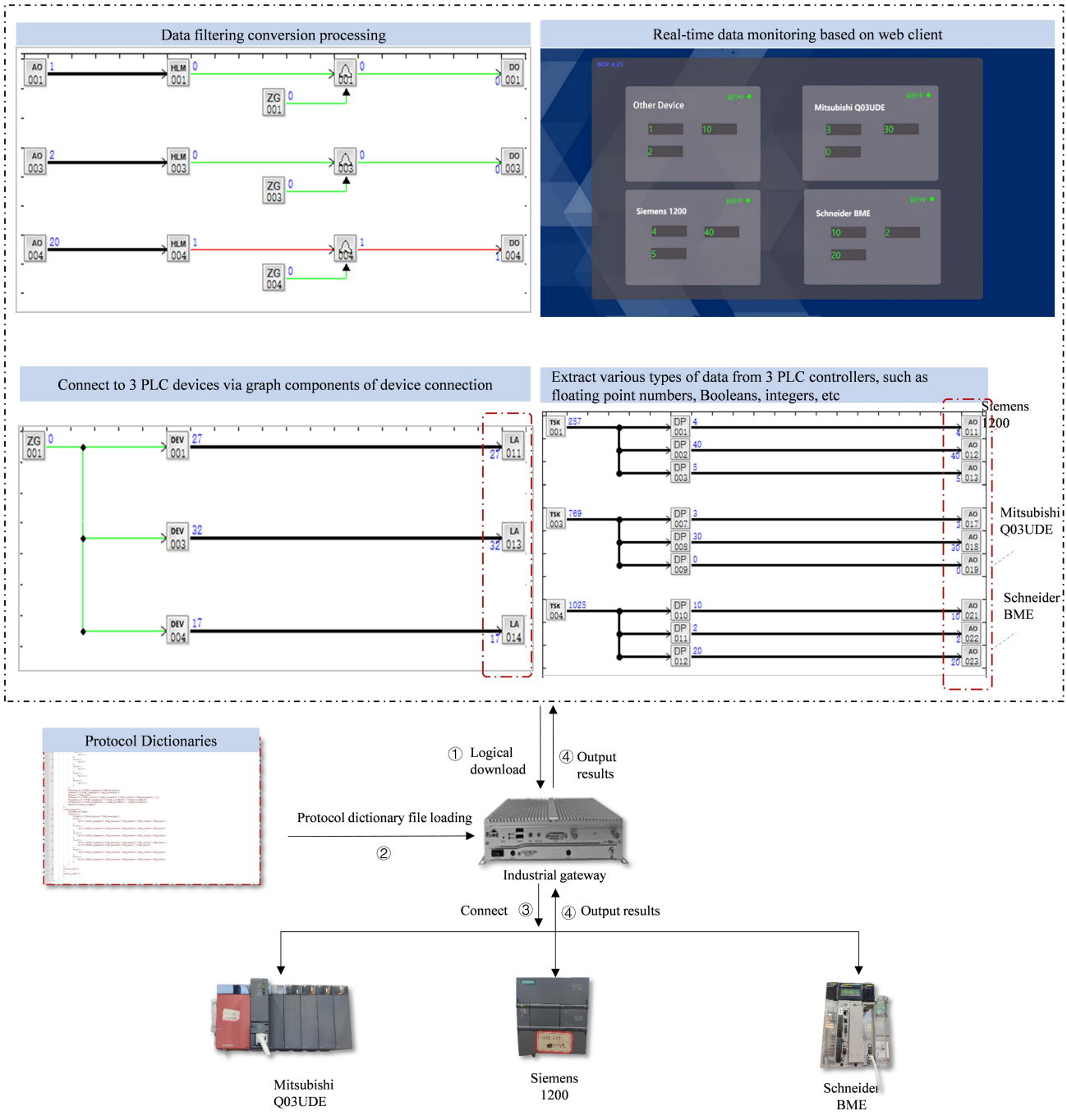


Figure 8. An example of data acquisition for heterogeneous device protocols.

The test results demonstrate that, based on a unified graphical component logic configuration program, the successful parsing and data conversion for the three types of PLC devices were achieved. The program accomplished the establishment of addressed connections with different types of PLCs, initiation of communication task requests, reading of data from specified memory areas, message parsing, extraction of multiple types of real-time data, and further computational processing such as base conversion and data filtering. By constructing algorithm programs for the MELSEC Communication Protocol (MELSEC)-QNA, S7-TCP, and Modbus-TCP protocols respectively, the graphical configuration system accurately parsed various data types including Boolean and Integer, and uniformly transmitted the data to the web-based visual monitoring interface, achieving centralized monitoring of data from heterogeneous devices.

To validate the accuracy of the gateway's protocol parsing, a data acquisition test was conducted targeting a Schneider PLC device supporting the Modbus-TCP protocol. During the test, the gateway read 10 data points at an acquisition frequency of twice per second, accumulating a total of 34,200 acquisition cycles. The experimental results (Table 5) indicate that the gateway achieved an average acquisition success rate of 99.98%, demonstrating its high-precision protocol parsing capability.

Table 5. Data acquisition success rate.

Request Parameters (Units)	Acquisition Cycle (ms)	Number of Acquisitions (Times)	Success Rate (%)
10	500	34,200	99.98

The data point capacity of a gateway is a crucial metric for evaluating its adaptability in complex industrial scenarios. To assess the performance of the interoperability gateway in concurrent data acquisition, data values with a length of 1024 were configured for each of the three memory areas (i.e., Coils, Input Registers and Holding Registers) within the Schneider PLC. Parsing pressure was applied utilizing multiple communication task graphical components. As shown in Figure 9, even when the number of data points was increased to 3000, the gateway accurately parsed the data, with data acquisition, database updating, and transmission functions all operating normally, demonstrating excellent scalability and processing capability.

In industrially complex processes, gateways must possess the ability for long-term stable operation. Therefore, this study further tested the gateway's continuous operational stability. A dynamic pulse wave simulation program (waveform period: 60 s) was embedded into the Schneider PLC, and the gateway's continuous operation time was calculated based on the pulse count. The experimental results (Figure 9) indicate that the interoperability gateway can operate stably for over 186 h continuously. Throughout this period, memory usage remained normal, data parsing and conversion were accurate and error-free, and no system crashes occurred, verifying its reliability in practical industrial environments.

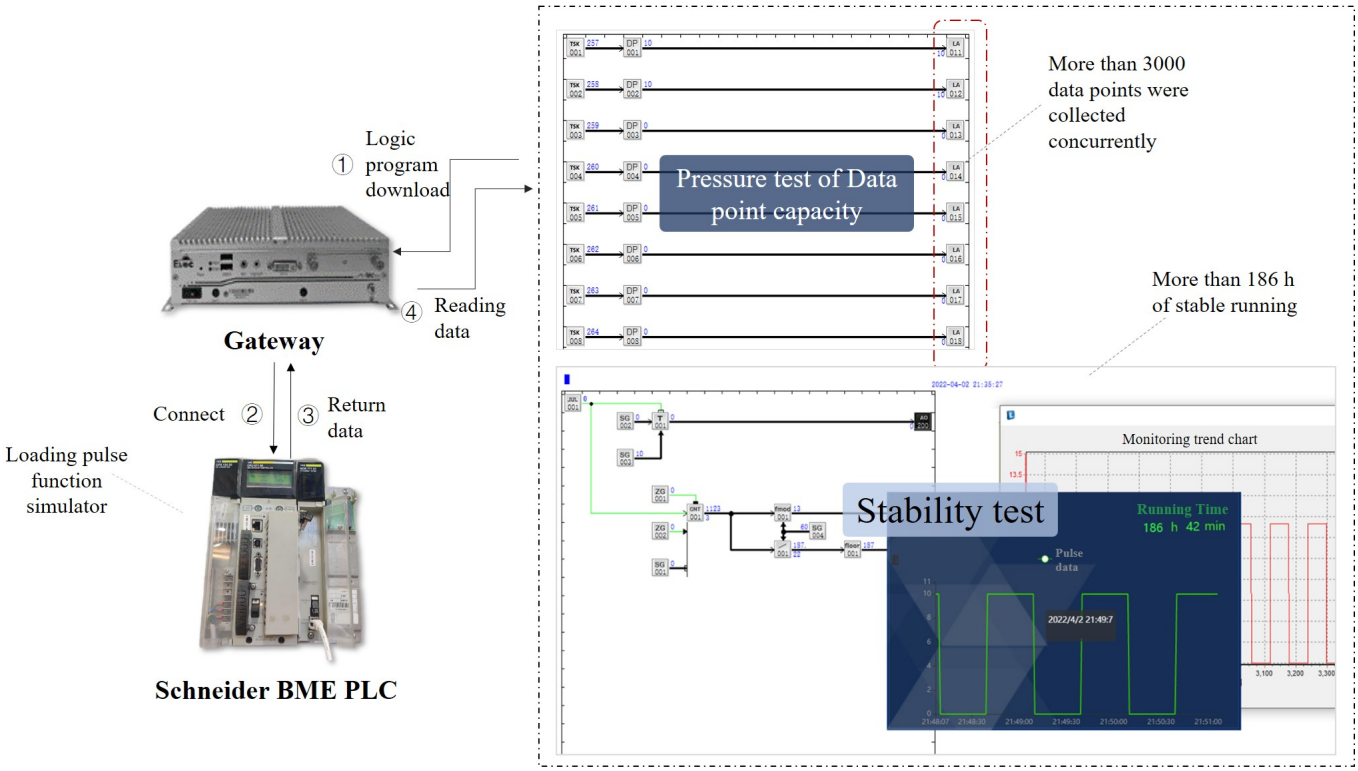


Figure 9. Gateway point capacity and stability testing.

In large-scale industrial systems such as SCADA, there are scenarios where multiple monitoring devices or historical data servers to concurrently access the gateway. To evaluate the gateway's performance in such a typical concurrent scenario, a targeted test was designed to simulate a condition where multiple historical servers simultaneously request large-volume data from the gateway. The test was configured with 9 concurrent users, each submitting an HTTP request to acquire 3000 parameters. The results (Table 6) indicate that under this load, the gateway achieved accurate protocol parsing with an error rate of 0%, a maximum response time of 169 ms, and a system throughput of 1.2 requests per second. Considering that SCADA systems typically employ a polling cycle of 1 to 2 s, the response latency of the proposed gateway is significantly lower than this threshold, demonstrating excellent real-time performance. Furthermore, its theoretical data processing capacity reaches 3600 points per second, which is fully capable of meeting the system's demand for second-level updates of a large number of data points. In summary, the gateway exhibits reliable real-time data processing capabilities in the target concurrent scenario.

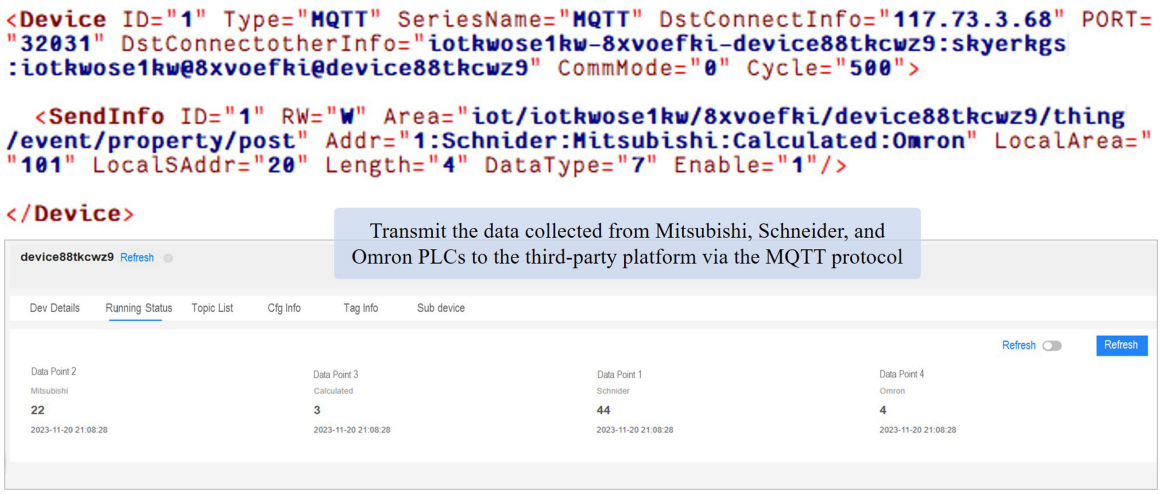
Table 6. Gateway concurrent load test.

Number of Concurrent Users	Request Parameters (Units)	Error Rate (%)	Maximum Response Time (ms)
9	3000	0	169

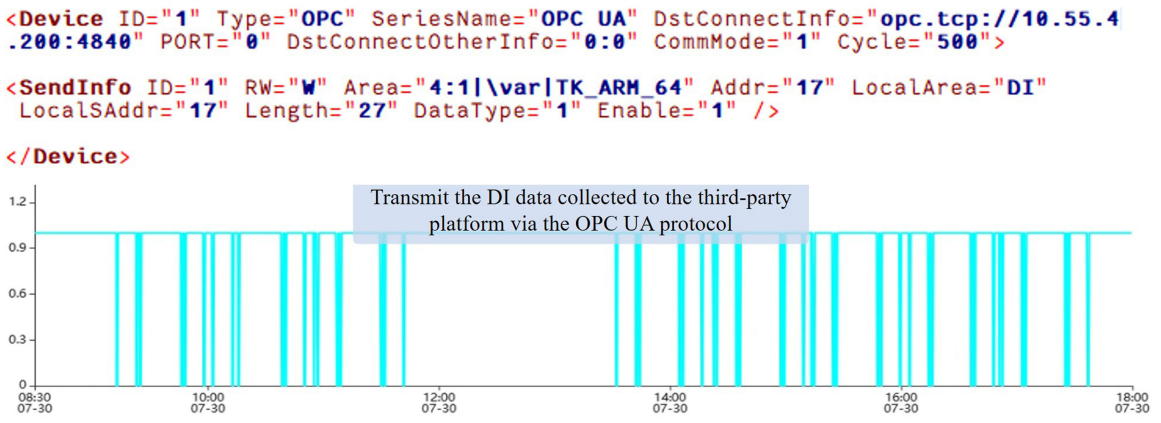
To evaluate the system's fault tolerance, a network sustained interruption experiment was designed to simulate hardware interface failures. During the test, the gateway's network port was deliberately disabled, forcing the system to operate offline for 172 h. Throughout this period, the gateway continuously cached 36 data points at a frequency of 5 s per point. After network connectivity was restored, the system successfully retransmitted the entire cache of data within 292 min, with data integrity fully maintained. The experimental results confirm

that the system maintains high reliability and data consistency even under extreme network anomaly conditions.

To validate the data integration capability of the GMGbox gateway with external systems, this study conducted systematic tests on its standard protocol interfaces. For the MQTT protocol test (Figure 10a), the gateway, acting as a client, successfully connected to a designated cloud server. Via XML configuration, it mapped heterogeneous data from Mitsubishi, Schneider, and Omron PLCs to a unified topic (thing/event/property/post), achieving the encapsulation and reliable transmission of multi-source device data. In the OPC UA protocol test (Figure 10b), the gateway, functioning as a client, connected to an independent OPC UA server and accurately wrote the acquired digital input (DI) data to the target variable node, verifying its interoperability at the industrial protocol level. The experiments demonstrate that the GMGbox gateway can efficiently convert acquired data into MQTT and OPC UA protocols through software-defined configuration and transmit it stably to third-party platforms, exhibiting excellent system compatibility and integration capabilities.



(**a**)



(**b**)

Figure 10. Gateway standard protocol interface tests: (**a**) MQTT client configuration and data transmission; (**b**) OPC UA client configuration and data transmission.

4.2. Case Study 1: Liquid Level Control System

To validate the data acquisition and integration capabilities of the proposed interoperability gateway in a typical industrial control scenario, this study designed and implemented a single-tank level control simulation system. Such systems are widely used in power generation, water treatment, and modern agriculture, where maintaining a stable liquid level is crucial for ensuring process continuity and efficiency. As shown in Figure 11, the system uses the tank level as the controlled variable and an electric inlet valve as the actuator, while introducing disturbances by adjusting the opening of a manual drain valve. Based on the deviation between the measured level from the sensor and the setpoint, a Proportional-Integral-Derivative (PID) controller dynamically adjusts the inlet valve opening to suppress or eliminate level fluctuations caused by the drain valve. The dynamic model of the tank and the PID control law are described by Equation (1) and Equation (2), respectively.

$$A\frac{dh}{dt} = Q_{in} - Q_{out} = K_1\mu_1 - K_2\mu_2\sqrt{h} \tag{1}$$

$$u(t) = k_p\left[e(t) + \frac{1}{T_i}\int_0^t e(t)dt + T_d\frac{de(t)}{dt}\right] \tag{2}$$

where Q_{in} and Q_{out} represent the real-time inlet and outlet flow rates of the water tank (m^3/min), K_1 and K_2 are the flow ratio coefficients of the electric valve and the manual valve (constants), μ_1 and μ_2 represent the opening coefficients of the electric valve and the manual valve, A and h denote the cross-sectional area (m^2) and the liquid level height (m) of the water tank, respectively. $u(t)$ is the output signal of the PID controller, k_p, T_i and T_d are the proportional, integral and derivative coefficients, respectively. $e(t)$ represents the control input, i.e., the deviation between the controlled variable and the setpoint.

The experimental system employed three heterogeneous PLCs (i.e., Omron NJ300, Mitsubishi Q03UDE, Schneider BME) to simulate the inlet valve control, drain valve control, and tank simulation unit, respectively, realistically recreating a scenario with multiple heterogeneous devices common in industrial settings. The gateway hardware utilized a low-cost embedded architecture, loaded with graphical modeling-based protocol parsing programs for Fins, MELSEC-QNA, and Modbus to acquire data from each PLC in real time and store it in a local database. The cloud-based web remote monitoring server retrieves data from the gateway database, tracks the current status of several key process variables, and performs continuous monitoring.

The experimental results (Figure 11) demonstrate that the graphical protocol parsing programs within the gateway accurately acquired parameters such as the inlet valve status, drain valve opening, and flow rates from the three PLCs. The web monitoring terminal, interacting with the gateway via cloud services, achieved real-time online monitoring of variables like the liquid level, valve openings, and flow rates. The entire data flow from PLC acquisition, through gateway transmission, to the cloud, and to the update of the monitoring interface exhibited a communication delay consistently below 100 ms. This meets the stringent real-time requirements of industrial field systems, effectively validating the feasibility and effectiveness of the proposed gateway solution in practical applications.

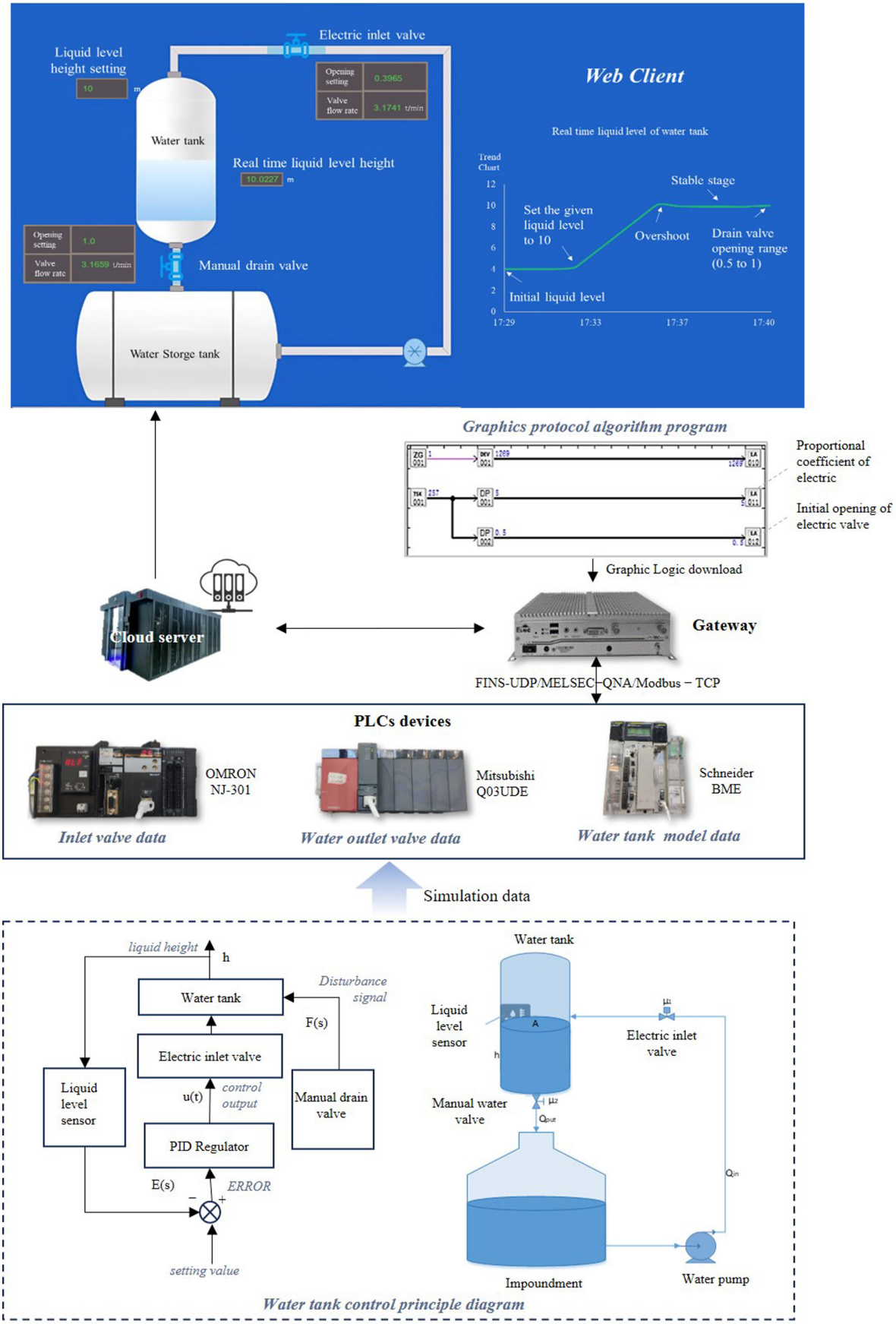


Figure 11. Application example of GMGbox in a liquid level control system.

4.3. Case Study 2: Application in a Wastewater Treatment SCADA System

To validate the effectiveness of the proposed interoperability gateway in a real-world industrial environment, this study deployed it within the SCADA system of a wastewater treatment plant. As shown in Figure 12, the x86 architecture gateway was tasked with integrating four Siemens S7-1500 series PLC controllers (via S7) deployed on-site and stably transmitting the acquired real-time data via its internal protocol and HTTP interface to the cloud-based SCADA monitoring system Client. Within this system, the gateway configured and transmitted over 10,000 data acquisition points, thereby providing a reliable data foundation for subsequent automated processing by cloud control systems and data visualization and analysis by the Manufacturing Execution System (MES). Field application results show that the gateway worked stably in the industrial environment for more than 90 days, with reliable data transmission. It can also seamlessly connect new devices through graphical configuration, effectively enhancing the SCADA system's capability to integrate and manage data from heterogeneous devices, which verifies its practical value in solving real engineering problems.

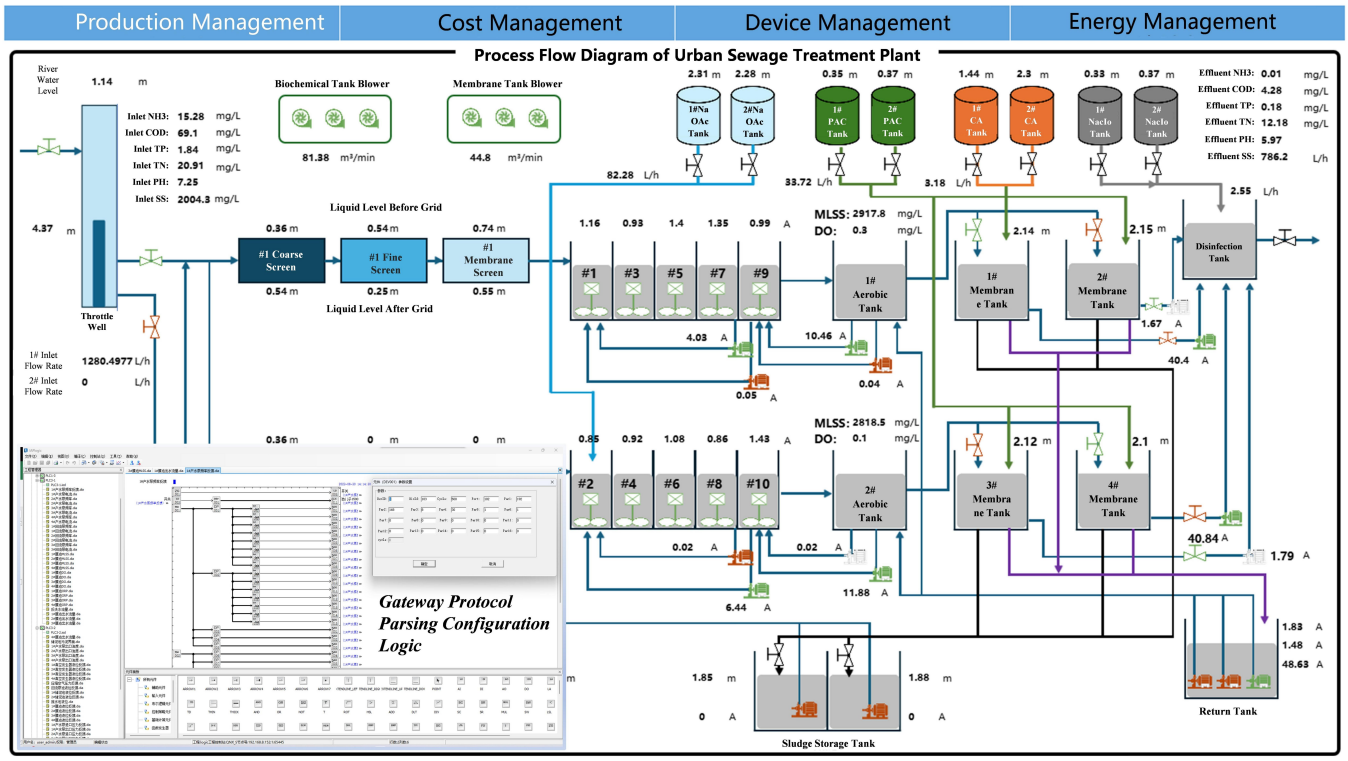


Figure 12. Application example of GMGbox in the SCADA System of a Sewage Treatment Plant.

5. Discussion and Evaluation

The core contribution of this study lies in proposing a novel architecture and paradigm that supports the integration of heterogeneous industrial protocols. To elucidate the value of this innovation, this section systematically analyzes the fundamental differences between the proposed method and existing graphical solutions (such as Node-RED) at the application level.

As shown in Table 7, the proposed GMGbox architecture demonstrates significant advantages over Node-RED in terms of decoupling strategy, protocol coverage, extensibility, and development and maintenance efficiency.

Unlike the general-purpose flow-based programming model adopted by Node-RED, GMGbox employs a hierarchically decoupled design oriented toward industrial commu-

nication semantics, where multiple modules including graphical components, protocol dictionaries, universal communication frameworks, and interface adapters collaborate in a division of labor. This architecture externalizes protocol structures, field definitions, and validation logic through graphical components and protocol dictionaries, achieving separation between logic and data.

Regarding protocol coverage, GMGbox natively supports mainstream industrial protocols and their variants such as S7, FINS, MELSEC-Q, Hostlink, and AB CIP. In contrast, Node-RED's protocol nodes primarily focus on IoT protocols like MQTT and HTTP, with support for industrial protocols largely dependent on third-party modules (e.g., Modbus, Ethernet/IP).

Table 7. Comparison of GMGbox and Node-RED in industrial protocol integration.

Comparison Item	Proposed GMGbox Method	Node-RED
Decoupling method	Graphical components, protocol dictionary, universal communication framework, interface adaptation	Flow-based orchestration layer, node-logic layer
Protocol coverage	Supports mainstream industrial protocols (S7, FINS, MELSEC-Q, Hostlink, AB CIP, etc.), specialized protocols and their variants	Supports IoT protocols (e.g., MQTT, HTTP); industrial protocols (e.g., Modbus, Ethernet/IP) are supported through community-contributed nodes or external libraries
Graphical node logic definition & granularity	Provides domain-specific graphical modeling language and components (e.g., Device Connection, Task Management, Data Parsing); protocol logic is dynamically defined via graphical components and protocol dictionary	Provides general-purpose communication nodes (e.g., TCP/UDP Read/Write, Function, Switch, Debug); Complex industrial frame parsing typically requires custom JavaScript functions or dedicated protocol nodes
Protocol variant extension mechanism	Universal parsing framework enables logic reuse; protocol types are extended by modifying graphical component parameters and protocol dictionary configurations	New or custom protocols are added via independent node packages or JavaScript-based integration; extending or adding new industrial protocols requires modifying multiple source files
Development & maintenance effort	Significant cost reduction achieved through reusable generic graphical components, externalized protocol dictionary, and industrial real-time monitoring/debugging tools	Developing new industrial protocol logic requires additional coding effort (e.g., frame parsing)

In terms of logical granularity, GMGbox provides semantic-level modeling components tailored for industrial communication scenarios (e.g., "Device Connection", "Task Management", "Data Parsing"), enabling direct definition of protocol logic through graphical configuration. Comparatively, Node-RED provides general-purpose flow orchestration (e.g., TCP read/write, Function, Switch), while complex or vendor-specific industrial protocols often require custom JavaScript or dedicated node development.

Regarding extensibility and reuse rate, GMGbox's universal protocol parsing framework minimizes modifications to underlying code, instead extending protocol types through dynamic adjustment of component parameters and updates to dictionary entries. Taking Modbus RTU/TCP as an example, the core tasks for protocol adaptation in GMGbox involve modifying connection parameters and dictionary entries, achieving a logic reuse rate exceeding 90%. In Node-RED, however, different protocol nodes operate independently, necessitating the creation of new handlers, adjustments to state machines,

and data frame parsing functions. Reuse occurs mainly at the code/library level, resulting in a relatively low rate of code reuse.

The above evaluation and analysis indicate that GMGbox can reduce development and maintenance effort by approximately 60% to 80% compared to Node-RED in typical industrial acquisition tasks. This significant difference primarily stems from its reusable generic components, externalized protocol logic, and support from unified debugging tools.

Therefore, the innovation of GMGbox is manifested not only in its graphical interface but, more importantly, in introducing an architectural methodology specifically designed for modeling heterogeneous industrial protocols. This approach elevates configuration tasks from low-level code manipulation to a higher level of semantic abstraction, enabling engineers to directly manipulate reusable components encapsulating communication semantics through a model-driven paradigm, thereby enhancing both protocol configuration efficiency and system debuggability.

6. Conclusions

This paper proposes a novel software-defined and graphical modeling paradigm for industrial interoperability gateways, aimed at addressing the core challenge of heterogeneous device integration in industrial control systems against the backdrop of Industry 4.0. The proposed GMGbox gateway middleware exhibits two salient features: firstly, the adoption of a layered architecture to decouple hardware resources from application functions; and secondly, the innovative use of graphical components to encapsulate protocol parsing and data conversion logic. The primary contributions of this research are threefold. First, a software-defined architecture was designed, featuring a layered and modular structure for the gateway software to provide a unified runtime environment for protocol parsing applications. Second, by decoupling data parsing and conversion functions, the traditional protocol driver model was refactored, enabling its definition through the codeless graphical configuration approach. Third, the GMGbox gateway middleware system, embodying the proposed paradigm, was designed and implemented and subsequently validated through systematic testing and engineering case studies.

Experimental results from system-level tests and case studies demonstrate the practicality and effectiveness of the proposed paradigm. During testing, the gateway successfully achieved concurrent parsing of multiple heterogeneous protocols by graphical configuration, supported stable communication for over 3000 data points, and exhibited reliable performance in continuous operation tests. Furthermore, its application in a level control system and a wastewater treatment plant SCADA system further validates its value in real-world industrial scenarios.

In conclusion, the novel gateway proposed in this paper establishes an architectural foundation for flexible industrial control by decoupling protocol logic from the underlying platform and embedding it in configurable graphical models.

Although this study primarily validated the paradigm's effectiveness through data acquisition from PLC devices, the proposed method is not limited to such devices or specific protocols. A current limitation is that protocol identification and debugging still rely on manual parameter configuration. Future work will focus on enhancing the gateway's intelligence, particularly in autonomous protocol identification, LLM-assisted parsing configuration, and parameter self-matching to address the challenges of integrating large-scale heterogeneous devices.

Author Contributions: Conceptualization, R.Z. and S.Z.; methodology, R.Z. and S.Z.; software, R.Z. and C.L.; validation, R.Z. and L.Y.; investigation, R.Z., L.Y. and H.W.; resources, S.Z. and C.L.; data curation, R.Z. and L.Y.; writing–original draft preparation, R.Z.; writing–review and editing, S.Z., C.L.,

L.Y. and H.W.; visualization, R.Z.; supervision, S.Z. and C.L.; project administration, S.Z.; funding acquisition, S.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Industrial Internet Innovation and Development Project of the Ministry of Industry and Information Technology (Grant No. TC19083WB) and the projects from Fujian Science and Technology Department (Grant No. 2024C0011 and 2024H0037).

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Acknowledgments: The authors would like to thank Songbin Chen, Huafeng Xiong and Ju Qiu from IAP Technology Co. for their valuable support in deploying the system and providing relevant experimental data and technical documentation.

Conflicts of Interest: Authors Rong Zheng, Chaoru Liu and Hongyu Wu were affiliated with the company IAP (Fujian) Technology Co., Ltd. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

APIs	Application Programming Interfaces
ARM	Advanced RISC Machines
CPU	Central Processing Unit
DDR3	Double Data Rate 3 Synchronous Dynamic Random-Access Memory
DP	Decentralized Periphery
EtherCAT	Ethernet for Control Automation Technology
EtherNet/IP	EtherNet Industrial Protocol
FMS	Flexible Manufacturing Systems
FINS	Factory Interface Network Service
Hostlink	Hostlink Communication Protocol
HTTP	Hypertext Transfer Protocol
ID	Identifier
IEC	International Electrotechnical Commission
IIoT	Industrial Internet of Things
IoT	Internet of Things
I/O	Input/Output
JSON	JavaScript Object Notation
LLM	Large Language Model
MELSEC	MELSEC Communication Protocol
MES	Manufacturing Execution System
MODBUS	MODBUS Communication Protocol
MQTT	Message Queuing Telemetry Transport
OPC UA	Open Platform Communications Unified Architecture
PID	Proportional-Integral-Derivative
PLCs	Programmable Logic Controllers
PROFIBUS	Process Field Bus Communication Protocol
PROFINET	Process Field Net Protocol
RJ45	Registered Jack 45
RMB	Renminbi
RS232	Recommended Standard 232
RS422	Recommended Standard 422
RS485	Recommended Standard 485
SAMA	Scientific Apparatus Makers Association
SCADA	Supervisory Control and Data Acquisition

S7	Siemens S7 Communication Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TSN	Time-Sensitive Networking
UDP	User Datagram Protocol
XML	eXtensible Markup Language
x86	Intel x86 Architecture

References

1. Wang, R.; Gu, C.; He, S.; Shi, Z.; Meng, W. An interoperable and flat Industrial Internet of Things architecture for low latency data collection in manufacturing systems. *J. Syst. Archit.* **2022**, *129*, 102631. [CrossRef]
2. Zhang, X.; Ming, X. A comprehensive industrial practice for Industrial Internet Platform (IIP): General model, reference architecture, and industrial verification. *Comput. Ind. Eng.* **2021**, *158*, 107426. [CrossRef]
3. Afrin, S.; Rafa, S.J.; Kabir, M.; Farah, T.; Alam, M.S.B.; Lameesa, A.; Ahmed, S.F.; Gandomi, A.H. Industrial Internet of Things: Implementations, challenges, and potential solutions across various industries. *Comput. Ind.* **2025**, *170*, 104317. [CrossRef]
4. Wang, P.; Luo, M. A digital twin-based big data virtual and real fusion learning reference framework supported by industrial internet towards smart manufacturing. *J. Manuf. Syst.* **2020**, *58*, 16–32. [CrossRef]
5. Cyberspace Administration of China. National Informatization Development Report (2024). 2025. Available online: https://www.cac.gov.cn/2025-07/30/c_1755590223034507.htm (accessed on 23 September 2025).
6. National Data Administration. Digital China Development Report (2024). 2025. Available online: https://www.nda.gov.cn/sjj/ywpd/sjzg/0530/ff808081-96b465bf-0197-200a135e-0536.pdf (accessed on 23 September 2025).
7. Jiang, Z.; Chang, Y.; Liu, X. Design of software-defined gateway for industrial interconnection. *J. Ind. Inf. Integr.* **2020**, *18*, 100130. [CrossRef]
8. Hazra, A.; Adhikari, M.; Amgoth, T.; Srirama, S.N. A Comprehensive Survey on Interoperability for IIoT: Taxonomy, Standards, and Future Directions. *ACM Comput. Surv.* **2021**, *55*, 1–35. [CrossRef]
9. Busboom, A. Automated generation of OPC UA information models—A review and outlook. *J. Ind. Inf. Integr.* **2024**, *39*, 100602. [CrossRef]
10. Ji, T.; Xu, X. Exploring the Integration of cloud manufacturing and cyber-physical systems in the era of industry 4.0—An OPC UA approach. *Robot. Comput.-Integr. Manuf.* **2025**, *93*, 102927. [CrossRef]
11. Trifonov, H.; Heffernan, D. OPC UA TSN: A next-generation network for Industry 4.0 and IIoT. *Int. J. Pervasive Comput. Commun.* **2023**, *19*, 386–411. [CrossRef]
12. Longo, E.; Redondi, A.E.C. Design and implementation of an advanced MQTT broker for distributed pub/sub scenarios. *Comput. Netw.* **2023**, *224*, 109601. [CrossRef]
13. De Rango, F.; Spina, M.G.; Iera, A. DLST-MQTT: Dynamic and lightweight security over topics MQTT. *Future Gener. Comput. Syst.* **2025**, *166*, 107625. [CrossRef]
14. Yrjölä, T. Using OPC Unified Architecture Information Models In PubSub Over MQTT Communication Model. Ph.D. Thesis, Aalto University School of Electrical Engineering, Espoo, Finland, 2024.
15. Liu, C.; Su, Z.; Xu, X.; Lu, Y. Service-oriented industrial internet of things gateway for cloud manufacturing. *Robot. Comput.-Integr. Manuf.* **2022**, *73*, 102217. [CrossRef]
16. AVEVA. PI System. 2025. Available online: https://www.aveva.com/en/products/aveva-pi-system/ (accessed on 23 September 2025).
17. SIEMENS. Insights Hub. 2025. Available online: https://plm.sw.siemens.com/en-US/insights-hub/ (accessed on 23 September 2025).
18. PTC. KEPServerEX. 2025. Available online: https://www.ptc.com/cn/products/kepware/kepserverex (accessed on 23 September 2025).
19. Barriga, J.A.; Clemente, P.J.; Pérez-Toledano, M.A.; Jurado-Málaga, E.; Hernández, J. Design, code generation and simulation of IoT environments with mobility devices by using model-driven development: SimulateIoT-Mobile. *Pervasive Mob. Comput.* **2023**, *89*, 101751. [CrossRef]
20. O'Keeffe, S.; O'Sullivan, D.; Bruton, K. Development of a modelling tool to aid the transition to carbon neutral industrial manufacturing. *J. Clean. Prod.* **2023**, *425*, 138604. [CrossRef]
21. Kang, G.; Cheng, H.; Liu, J.; Wen, Y.; Peng, J. Business Process Modeling for Industrial Internet Application via BPMN Extension. *IEEE Trans. Autom. Sci. Eng.* **2025**, *22*, 813–829. [CrossRef]
22. Neis, P.; Wehrmeister, M.A.; Mendes, M.F.; Pesente, J.R. Applying a model-driven approach to the development of power plant SCADA/EMS software. *Int. J. Electr. Power Energy Syst.* **2023**, *153*, 109336. [CrossRef]
23. Rosiak, K.; Schlie, A.; Linsbauer, L.; Vogel-Heuser, B.; Schaefer, I. Custom-tailored clone detection for IEC 61131-3 programming languages. *J. Syst. Softw.* **2021**, *182*, 111070. [CrossRef]

24. Measurement, Control & Automation Association (MCAA). Functional Diagramming of Instrument and Control Systems: Analog and Digital Control Systems. 1981. Available online: https://themcaa.org/wp-content/uploads/functionaldiagramming.pdf (accessed on 23 September 2025).
25. Node-RED Team. Node-RED Documentation. 2025. Available online: https://nodered.org/docs (accessed on 23 September 2025).
26. *IEC 62541-1*; OPC Unified Architecture—Part 1: Overview and Concepts. International Electrotechnical Commission (IEC): Geneva, Switzerland, 2010.
27. Xu, J.; Fang, Y.J. Development of embedded fieldbus protocol conversion gateway. *Comput. Eng.* **2006**, *32*, 255–257.
28. Calderón, D.; Folgado, F.J.; González, I.; Calderón, A.J. Implementation and Experimental Application of Industrial IoT Architecture Using Automation and IoT Hardware/Software. *Sensors* **2024**, *24*, 8074. [CrossRef]
29. Nițulescu, I.-V.; Korodi, A. Supervisory Control and Data Acquisition Approach in Node-RED: Application and Discussions. *IoT* **2020**, *1*, 76–91. [CrossRef]
30. Trunzer, E.; Wullenweber, A.; Vogel-Heuser, B. Graphical modeling notation for data collection and analysis architectures in cyber-physical systems of systems. *J. Ind. Inf. Integr.* **2020**, *19*, 100155. [CrossRef]
31. Zheng, S.; Zhang, Q.; Zheng, R.; Huang, B.; Song, Y.; Chen, X. Combining a Multi-Agent System and Communication Middleware for Smart Home Control: A Universal Control Platform Architecture. *Sensors* **2017**, *17*, 2135. [CrossRef] [PubMed]
32. Zheng, S.; Ni, W.D. Research and Implementation of Dynamic Reconfiguration Technology in Distributed Control System. *At. Energy Sci. Technol.* **2009**, *43*, 724–729.
33. Zheng, S.; Lin, Z.; Zeng, Q.; Zheng, R.; Liu, C.; Xiong, H. IAPcloud: A Cloud Control Platform for Heterogeneous Robots. *IEEE Access* **2018**, *6*, 30577–30591. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.