

Article

IoT OS Platform: Software Infrastructure for Next-Gen Industrial IoT

Zhihuan Xing ^{1,*}, Yuqing Lan ², Zhongjie Sun ^{2,3}, Xiaoyi Yang ², Han Zheng ^{2,3}, Yichun Yu ^{1,*}  and Dan Yu ³¹ School of Computer Science and Engineering, Beihang University, Beijing 100191, China² School of Software, Beihang University, Beijing 100191, China³ China Standard Intelligent Security Co., Ltd., Beijing 100097, China;

* Correspondence: xingzhihuan@buaa.edu.cn (Z.X.); yichunyu@buaa.edu.cn (Y.Y.)

Abstract: With the rapid development of the Internet of Things (IoT), the growth of the Industrial Internet of Things (IIoT) applied in the industrial sector has also been swift. However, in practical applications, there are still issues such as the misalignment between theory and application, the lack of a unified standardization framework, and the frequent occurrence of data silos. These issues limit the maintainability and scalability of IIoT systems and increase the digitalization costs for enterprises. Based on this, drawing from the design principles of classic general-purpose operating systems, we propose the concept of an IoT operating system platform. As a software infrastructure aimed at the next generation of IIoT, the IoT OS platform consists of general-purpose computer operating systems and the platform software running on them. It manages computing resources and entities such as sensors, networks, and ubiquitous artificial intelligence applications or systems, and provides service support for IIoT applications upwards, aiming to improve existing issues and enhance the specificity and scalability of IIoT systems. This paper presents the current status of IIoT systems, the definition and architecture of the IoT OS platform, and validates the theoretical architecture through specific cases.

Keywords: IoT; industrial IoT; IoT system; IoT operating system platform



Citation: Xing, Z.; Lan, Y.; Sun, Z.; Yang, X.; Zheng, H.; Yu, Y.; Yu, D. IoT OS Platform: Software Infrastructure for Next-Gen Industrial IoT. *Appl. Sci.* **2024**, *14*, 5370. <https://doi.org/10.3390/app14135370>

Academic Editor: Andrea Prati

Received: 25 April 2024

Revised: 8 June 2024

Accepted: 20 June 2024

Published: 21 June 2024



Copyright: © 2024 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 concept of the Internet of Things (IoT) was proposed by Ashton [1] and formally established in 2005 [2], and has been developing for over two decades. Compared with other new-generation information technologies such as big data, cloud computing, and artificial intelligence, the IoT presents a characteristic where “applications precede theory”: the widely recognized three-layer IoT architecture (perception layer, transport layer, and application layer) is too generalized and insufficient to guide practice; subsequent IoT architectures proposed are limited to the interpretation of existing applications in some industries and fail to reach a consensus within the industry [3].

Industrial IoT (IIoT) is a key area of IoT application. By applying IoT technology in the industrial field, combining connection, perception, analysis, and control methods, it achieves intelligence, automation, and efficiency in industrial production [4–6]. In various scenarios of the IIoT, the architecture design of IoT systems is often specialized for a particular project, lacking generalization standards [6–8], which affects the maintainability and scalability of IIoT systems [3]. Traditional IIoT systems usually can only meet the existing needs for a short time after project delivery and cannot cope with new demands, unless by building a new IoT system to meet new needs [9]. The recurrence of the above situation has led to the formation of data silos among various IoT systems. Lower-level IoT systems are unable to integrate all the data perceived by physical objects in a unified manner, affecting the upper-level analysis and decision-making, increasing the cost of enterprise digitalization, and thus slowing down the pace of industrial digitalization [10,11].

To meet the construction needs of IIoT systems and avoid the predicament of traditional IoT system construction, we propose the concept of an IoT operating system platform. As a software infrastructure for the next generation of IIoT, the IoT OS platform follows the design philosophy of classic operating systems, positioning the platform as the core of the IIoT system: managing various entities downwards and providing service support for IIoT applications upwards. This platform can improve the disadvantages of traditional IoT system design, cater to the needs of industrial industry digital transformation, and deliver IIoT systems with greater specificity and scalability.

The remainder of this article is organized as follows: Section 2 elaborates on the related work and current research status of IIoT system construction. Section 3 presents the definition and specific architecture of the IoT operating system platform. Section 4 introduces how to carry out IIoT construction based on the IoT operating system platform through case studies. Section 5 is a summary and outlook of the IoT operating system platform.

2. Related Work and Key Challenges

2.1. Industrial IoT System Architecture

In the research on IoT systems, the classic three-layer architecture divides IoT systems into perception, network, and application layers. These are responsible for sensing and collecting data, connecting with other physical entities or systems, and providing end-to-end applications to users, respectively [12–16]. Tan proposed a middleware-based architecture, isolating middleware as a separate layer and defining a coordination layer to handle the standardization of data [17,18]. R. Khan's five-layer architecture also focuses on middleware, abstracting a processing layer composed of middleware specifically for connecting, organizing data, providing basic capabilities, and abstracting a business layer to include attention to actual business elements such as profit models and user privacy [16,19]. There are also architectures with different specific layer definitions but similar concepts, such as the three-layer architecture described by Xu [20], and the four-layer architecture described by Shu [21]. With the introduction of fog computing (or edge computing) concepts [16,22,23], architectures have increasingly emphasized the concept of cloud-edge collaboration, introducing elements such as edge gateways and edge computing nodes [12,24]. Some architectures incorporate sociological theories, defining basic components such as ID, metadata, service discovery, and studying objects in physical network systems as nodes in social networks [16,25].

As a subset of IoT systems, IIoT systems also need to face devices with limited resources and networks with limited transmission capacity. Compared to ordinary IoT systems, IIoT systems place more emphasis on machine-to-machine communication and coordination, and deal with larger volumes of data, requiring stricter standards for latency, jitter, reliability, etc. [20,24,26]. Germany's Electro and Digital Industry (ZVEI)'s Reference Architectural Model Industrie 4.0 (RAMI 4.0) presents a six-layer architecture: asset, integration, communication, information, functional, and business layers. It interprets IIoT systems by combining hierarchical architecture as one dimension with elements such as product lifecycle and system control segments [27,28]. The Industry Internet Consortium (IIC) released a reference architecture document that introduces the issues that IIoT systems should focus on from a business viewpoint, usage viewpoint, functional viewpoint, and implementation viewpoint [29] and provides a specific system architecture.

2.2. Key Challenges

We categorize the typical challenges encountered in constructing IIoT systems into four aspects: heterogeneity, real-time, intelligence, and security.

2.2.1. Heterogeneity

In the practical implementation of IIoT, heterogeneity is an unavoidable issue. Beyond the commonly discussed heterogeneity of underlying protocols, a greater challenge comes from the integration of physical devices and production systems, with their business model

heterogeneity. Although solutions like LwM2M, OPC UA attempt to resolve the heterogeneity of both the underlying protocols and business models by defining information models at the application layer [30–32], the widespread adoption of unified standards in production environments is often a prolonged process [10], and does not make old production equipment and systems compatible with new protocols. As various manufacturers have invested substantial resources in their complete hardware and software ecosystems, they are reluctant to adopt a new standard that might harm their interests, making heterogeneity a restraining factor in the development of IIoT systems.

2.2.2. Real-Time

Currently, the basic network topology of many IIoT systems is star-shaped, meaning devices connect in a single hop to a central IoT platform or to a gateway, which then connects to the central IoT platform [33]. In such topologies, any data transmission requires support from the central IoT platform, leading to network transmission latency and data processing latency. Industrial monitoring applications are not sensitive to latency, but control applications have high real-time requirements, only tolerating millisecond-level delays [24]. IIoT systems introduce edge computing nodes to perform some control and data analysis work closer to the devices, thereby reducing transmission and processing latency [4,23,34]. However, in complex scenarios, the deployment of edge nodes becomes difficult, and there are no systematic solutions to issues like network switching, data transmission, computing task distribution, and result acquisition [35]. Furthermore, some research points out that a key issue in edge computing is addressing the latency caused by physical distance in data access [36].

2.2.3. Intelligence

In IIoT scenarios, various sensing devices continuously produce a large amount of time-series data, and extracting value from such data requires the introduction of artificial intelligence, machine learning, etc., to support intelligent optimization and decision-making in IIoT systems. Different types of devices provide different data, first requiring different schemes for data cleaning to ensure data integrity and standardization [37]. Further, a wide variety of analysis algorithms are needed for different IoT application scenarios, such as face recognition, defect detection visual models [38,39], log processing natural language models [40], or anomaly detection, predictive analysis structured data processing models [41,42], each with unique development processes, integration methods, and usage. Finally, various intelligent algorithms need to work on the central side or edge side, facing a variety of hardware platforms and resource conditions, requiring complex porting processes [4,34]. Therefore, acquiring and integrating algorithms to intelligize the system is a major challenge for IIoT systems.

2.2.4. Security

Currently, in the field of IoT security, there has been extensive research and application of technologies such as asymmetric encryption, certificates, access control, redundancy backup, and blockchain [43,44]. However, such security protection strategies usually only focus on the security of the underlying technology stack. Considering that the process of digital transformation in IIoT often requires the integration of various existing systems, the complexity of business integration between systems far exceeds that of the traditional internet. There is still a lack of systematic solutions to the business security issues brought about by information transmission between entities in IIoT systems.

3. Theoretical Framework of the IoT OS Platform

Recognizing the development needs of the IIoT and addressing the key problems in the practical implementation of IIoT technologies, we propose a theoretical framework for the IoT OS platform. This framework aims to guide the architectural design and system construction of next-generation IIoT solutions scientifically.

The IoT OS platform, serving as the software infrastructure for IIoT systems, manages various entities downwards and provides services support to IIoT applications upwards. Based on the operating system concept, it extends its computing resource management capabilities outward, moving from traditional operating systems' management of local computing resources to include any entity that can participate in the construction of IoT applications across multiple dimensions, such as sensor resources, network resources, and software resources. Its core philosophy can be divided into five aspects: connectivity, real-time, intelligence, openness, and security, thereby supporting various applications at the upper layer of the IIoT.

3.1. Structured Organization Mechanism for IIoT

To build the theoretical framework of the IoT OS platform, it is first necessary to research the structured organization mechanism of the Industrial Internet. Hence, this paper initially proposes a five-layer architectural model based on the IoT OS platform for the IIoT, including perception layer, transport layer, platform layer, application layer, and business layer, scientifically and effectively dividing and characterizing all elements of the IIoT.

3.1.1. Perception Layer

The perception layer encompasses entities within IIoT scenarios that receive or transmit data. The most common objects at the perception layer in industrial scenarios are sensors that convert physical information into data for upward transmission and actuators that receive data and convert it into physical actions. In the context of information transformation in the industrial sector, most target enterprises have already achieved some level of digitalization, and do not need to deploy a new set of sensors or actuators [10]. Thus, the upper layer must have the capability to access and integrate with the existing digital achievements of target enterprises. Therefore, the perception layer in IIoT scenarios should also include databases, edge computing devices, informatization systems, and other entities commonly found in industrial informatization scenarios [10].

3.1.2. Transport Layer

The transport layer describes the various protocols involved in data transmission within IIoT scenarios. With the advancement of communication technologies, some protocols can no longer be categorized into a single layer of the network protocol stack but span multiple layers or coexist with peer layer protocols [17]. To describe data transmission in IIoT scenarios, the transport layer in this architecture is divided according to different business types, thus better expressing the flexible orchestration and combination of protocols under limited transmission conditions in IIoT scenarios. Application protocols, such as CoAP, MQTT, HTTP, AMQP, OPC UA, Modbus, etc., are a category of protocols that describe data transmission businesses, including different degrees of implementation for establishing connections, transmitting data, and parsing data. Service discovery protocols are crucial for achieving interoperability among IIoT devices, similar to DNS in the traditional internet. Supporting service discovery protocols enables self-organizing networks and interoperability of IIoT devices based on the IoT OS platform. Infrastructure protocols [17] manage communication and data exchange between different components of the network infrastructure, ensuring reliable and efficient data transmission. Security protocols are a focal point in IIoT scenarios [17], constructing an efficient and reliable security protocol stack with the support of the IoT operating system platform to address data transmission and application among a massive number of heterogeneous devices in IIoT scenarios.

3.1.3. Platform Layer

As previously defined, the IoT OS platform's role is to manage various entities downwards and provide service support for IIoT applications upwards. By implementing core access capabilities, the platform layer can act as a key infrastructure for data carrying.

However, naive IoT access capabilities are not sufficient to support the business needs of an entire IIoT solution. Currently, under the push of Industry 4.0, more scenarios are presenting application demands for intelligence, security, real-time, etc. As the carrier of IIoT applications, the design style and capability boundaries of the platform layer directly affect the quality of the overall solution.

3.1.4. Application Layer

The application layer focuses on technically describing various IoT applications. This division method for the application layer reveals the application support capabilities required in IIoT scenarios by the platform layer. Currently, technologies like big data, artificial intelligence, digital twins, automation, and cloud-edge collaboration have relatively well-developed theoretical support but lack sufficient practical implementation in IIoT scenarios [7]. Additionally, the bottleneck in the digital and intelligent transformation of the industry lies in the high barriers and costs of constructing intelligent applications, making it difficult to achieve the expected intelligent goals [7]. In some studies, the application layer is divided according to forms such as predictive maintenance, electric power inspection, etc. [45]. However, this classification method cannot comprehensively cover all application scenarios, as industry applications are always emerging. But, technology is limited in the foreseeable future; dividing IIoT applications based on the technology stack can better guide system design, thereby addressing the gradual digital transformation in the industry.

3.1.5. Business Layer

The business layer describes representative vertical industries in the IIoT, which have urgent demands for intelligent transformation [5]. These industries require efficient data collection, real-time monitoring, and intelligent decision-making to improve production efficiency, reduce costs, enhance security, and increase reliability [10]. The IoT OS platform constructs industry-specific IIoT applications with a standardized architecture, further combining applications according to needs to form industry-specific closed-loop solutions, achieving intelligent transformation of traditional industrial sectors.

3.2. Core Design of the IoT Operating System Platform

The theoretical design of the IoT operating system platform starts with the four most critical challenges: heterogeneity, real-time, intelligence, and security, aligning with the general capabilities of other IoT platforms [46]. Ultimately, our theoretical architecture is as shown in Figure 1.

3.2.1. Connectivity

Based on the direction of data flow, we categorize heterogeneous devices in the IoT into sensors and actuators. Sensors are devices that sense data and send it to the platform, while actuators respond to platform data. The ability to connect sensors and actuators is a foundational capability of the IoT OS platform. This requires the platform to have the capability of pluggable custom protocol parsing methods and adaptive matching communication methods, to access the devices of different communication methods, protocols, and brands. Supporting different protocols is a prerequisite for managing heterogeneous devices and is also one of the foundational capabilities of the IoT OS platform. This capability can improve device compatibility, reduce development difficulty, simplify management operations, and enhance communication efficiency and reliability, thereby achieving interoperability between devices. However, the IoT features a variety of transmission protocols and protocol carriers. Unlike internet transmission protocols, which focus on standardization and high speed, IoT applications prioritize transmission stability, reliability, and real-time performance [17], requiring transmission can still operate normally in relatively harsh environments. Additionally, IoT devices often lack computing resources and have limited transmission capabilities. In addition to resource-rich devices that support TCP/IP-based network protocols, many IoT devices only support field protocols [17].

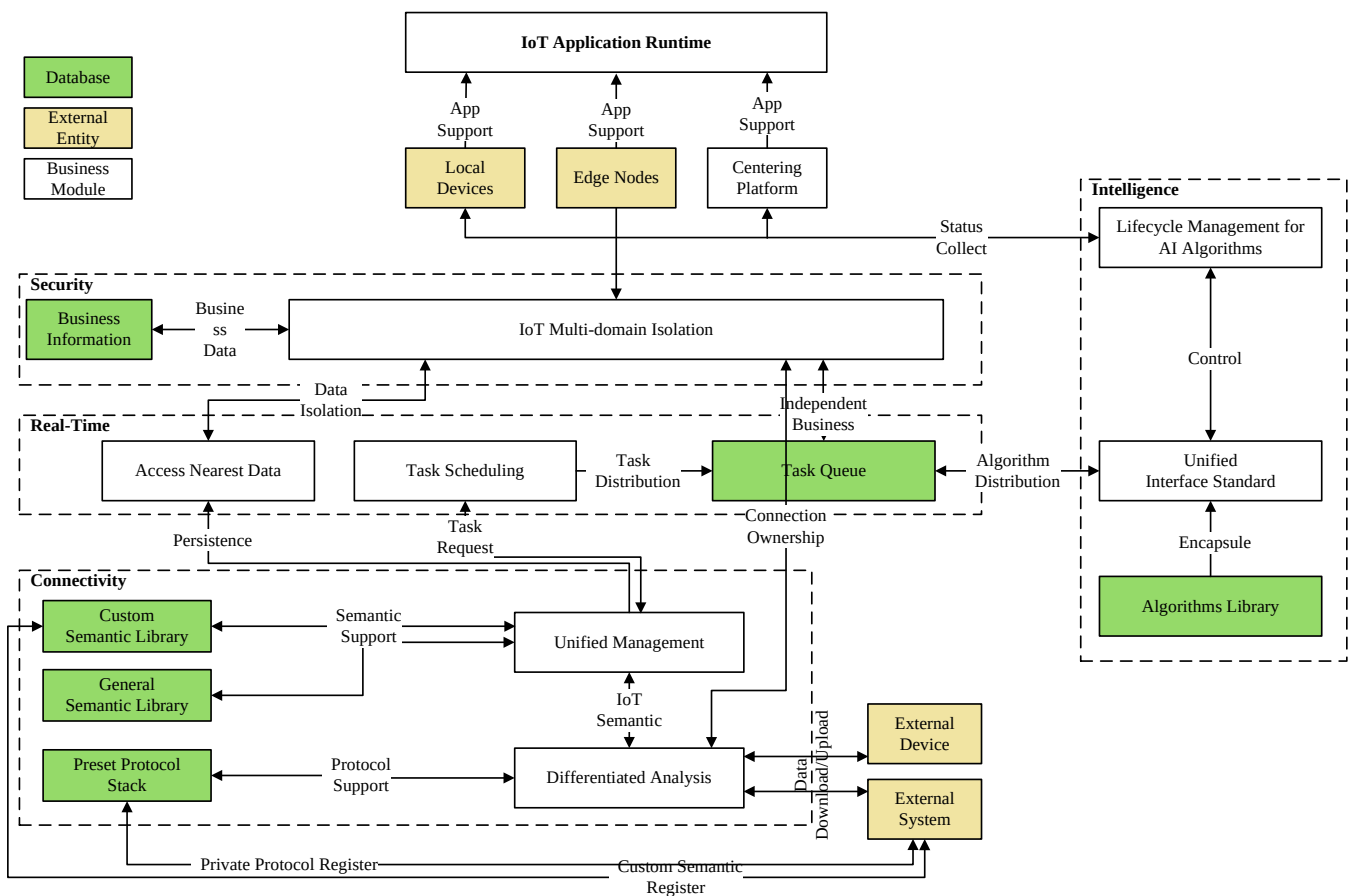


Figure 1. Theoretical architecture of IoT OS platform.

The connectivity capability of the IoT operating system platform parallels the device management capability of traditional operating systems. In traditional operating systems, device management is used to uniformly manage and control external I/O devices. Considering the diversity and differentiation in IIoT scenarios, in the theoretical framework of the IoT OS platform, we generalize the definition of I/O devices to physical entities capable of data reading and writing (such as sensors and actuators) or abstract informatized entities representing physical entities (such as external systems or databases). In traditional IoT system construction, there is a lack of unified, standardized devices and they usually cannot integrate with existing informatization systems. This prevents the reuse of previous digital achievements when advancing industrial digital construction, and newly constructed IoT systems cannot retain forward compatibility for future new demands [47].

Inspired by the architecture of traditional operating systems, the theoretical framework of the IoT OS platform constructs connectivity capabilities, shielding the heterogeneity and differentiation of lower-layer entities from upper-layer applications. Through differentiated analysis and unified management, we achieve entity management with forward compatibility, as illustrated in Figure 2.

Differentiated Analysis. In traditional operating systems, to achieve complete interaction with external I/O devices, it is necessary to build device drivers targeting differentiated external I/O devices, acting as a bridge between the operating system and hardware devices, converting communications between the operating system and hardware devices into control signals required by the devices [48]. Similarly, in the IoT OS platform, interacting with differentiated external entities also requires building specific data parsing functions. Although IIoT entities lack a unified standard, they all fundamentally involve data I/O based on bits or bytes [49]. Currently, entities involved in IIoT scenarios have two characteristics: the enumerability of network connection protocols and the innumer-

ability of specific semantics. Considering that communication is composed of composite network protocol stacks, most protocols can be pre-integrated through open standards at the technical level. Some private or unknown protocols are also built upon known protocols, integrating known protocol stacks and unknown protocol lower network stacks to build complete support capabilities for network connection protocols. The semantics of existing IIoT devices often exhibit significant differences [50], making it impractical to extract a unified semantic standard at the entity level without a specific business context.

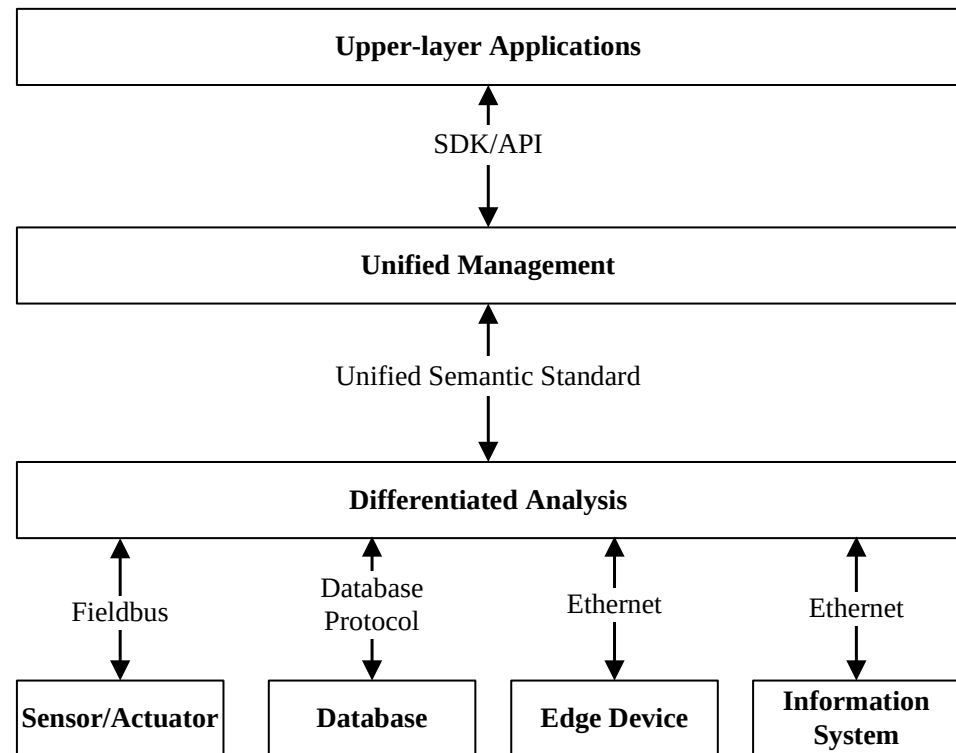


Figure 2. Connectivity capability of IoT OS platform.

Therefore, the IoT OS platform allows for establishing network connections based on any network layer with known protocols, opening up customized development capabilities for specific protocols and semantics for data I/O, thus completing differentiated analysis capabilities for external entities. By integrating existing network protocol stacks, development costs are minimized, and rapid integration with upper-layer business applications is achieved.

Unified Management. After completing entity access to the platform, the next step is to manage the entities using a unified semantic, enabling interoperability between upper-layer businesses and entities, as well as among entities. The biggest obstacle to achieving interoperability between entities in actual IIoT scenarios is semantic differentiation [50].

By constructing a unified semantic standard on the IoT operating system platform, leveraging the platform's pervasive access capability, entities based on the IoT operating system platform can communicate in a unified language. Since the IoT operating system platform has already shielded the protocol differences of lower-layer entities, interoperability between entities and between entities and upper-layer services only requires using the unified semantic standard provided by the IoT operating system platform.

Specifically, the semantic standard provided by the IoT operating system platform can be divided into general semantics and customized semantics, as shown in Figure 3. General semantics describe the basic capabilities of any entity at an abstract level, such as open, close, read, write, exception, etc. Customized semantics, on the other hand, are common and self-explanatory functional interfaces agreed upon between entities according to actual

business needs. These functional interfaces are uniformly managed by the IoT operating system platform, thereby achieving full coverage of semantic standards for business needs.



Figure 3. Semantic standards.

3.2.2. Real-Time

In IIoT scenarios, there are issues with large volumes of data, high real-time requirements, and significant security risks. Various attempts have been made domestically and internationally to address these issues, mainly by introducing and adapting technologies from other fields with characteristics of the IIoT industry. To solve problems brought by massive data, the consensus is that single-machine systems are no longer sufficient, and the introduction of distributed systems is necessary [51]. Many studies have been conducted from the perspectives of communication and wireless networking, focusing on improving the underlying network for real-time data transmission, but lacking attention to business real-time requirements. Existing IIoT systems often integrate multiple terminal devices and systems. In their distributed network architecture, adopting decentralized data access and business execution strategies can enhance real-time performance in more business scenarios.

To meet the real-time requirements in the context of massive data in IIoT, we have designed mechanisms for nearest data access and task scheduling in the IoT operating system platform.

Nearest Data Access. In operating systems, data are stored in high-speed caches, main memory, or auxiliary storage according to access frequency, in order to enhance data access speed and storage capacity. The data access model of the IoT operating system platform is as shown in Figure 4.

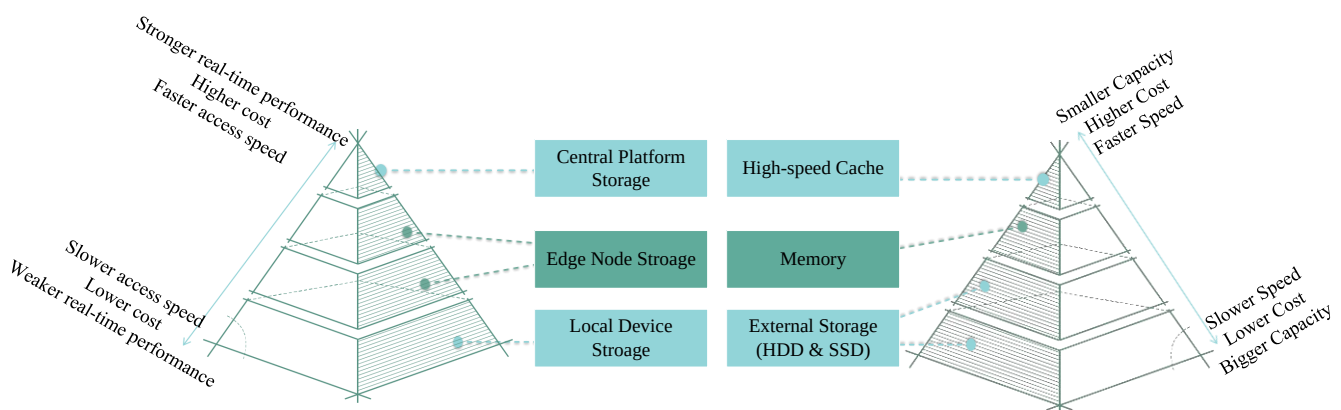


Figure 4. Model of data access.

Specifically, when storing data, we designed a mechanism that allows the IoT OS platform to schedule data across cloud, edge, and end sides based on the importance and access frequency of the data. This enables better support for data access speed and processing efficiency in scenarios with massive data, reducing data storage costs. We primarily consider the data volume and real-time requirements in evaluating data storage. High-frequency data and important alarm data, such as device attributes and device alarms, should be stored in the central platform for rapid access when needed. For data of lesser importance and weaker real-time requirements, they should be stored in edge nodes and

accessed when needed, such as audio and video data, which are usually transmitted to the platform when users stream. For low-frequency data, such as device authentication information and signaling logs, they are stored directly in devices and not uploaded to edge nodes or the central platform, reducing storage overhead and communication costs.

Task Scheduling. In complex IIoT scenarios, IoT systems need to execute a wide variety of tasks. Different tasks have different resource requirements. Some tasks have real-time requirements and need to be executed on edge devices or edge servers, while others require significant computational power and need to be handed over to central cloud nodes. To meet the resource demands of different tasks, we believe that IIoT systems need the ability to dynamically and reasonably allocate resources and schedule tasks.

In operating systems, the execution of processes occupies CPU time, memory space, files, network ports, etc. Based on factors such as process priority and execution time, the operating system can schedule and manage resources. In our design for the IoT OS platform, computational resources and storage resources are unevenly distributed across the distributed IIoT system. Many services provided to the upper layer also require the coordinated scheduling of various resources, including network resource coordination during transmission, data resource coordination during data flow, and computational resource coordination during task execution. Referring to traditional operating system resource allocation strategies, we designed a unified resource allocation system in the IoT OS platform, as shown in Figure 5, to collect real-time information from each subsystem. Based on factors such as task priority, the system implements the optimal configuration of various types of resources in various task scheduling scenarios.

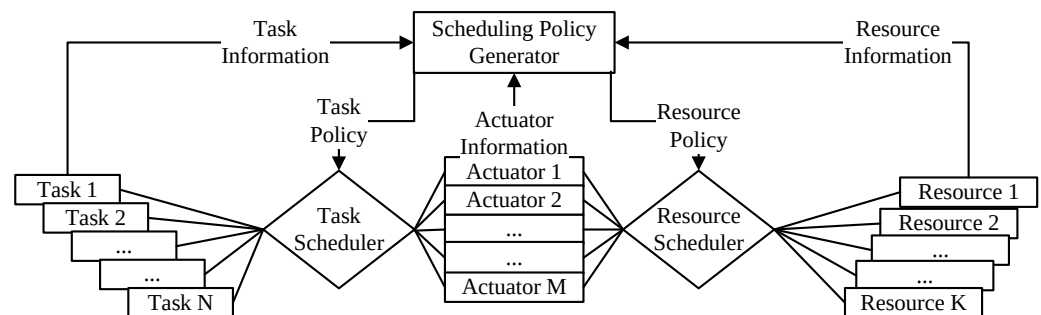


Figure 5. Task scheduling system.

Task Offloading. Moreover, inspired by the process scheduling concept in operating systems, where multiple processes share computational resources, the operating system selects scheduling strategies such as round-robin, priority scheduling, multilevel feedback queue scheduling, and shortest process first scheduling, based on the targeted scenarios and process task types. This achieves fair competition among processes, enhancing the system's resource utilization and performance. We constructed a task offloading mechanism, as shown in Figure 6, allowing the IoT OS platform to decompose tasks and distribute task slices to queues on the cloud-edge-end side.

The computing results are then collected at the central server for aggregation. Utilizing the aforementioned design enables the IoT OS platform to intelligently schedule tasks among the distributed architecture in complex IIoT business scenarios, optimizing the overall system execution efficiency.

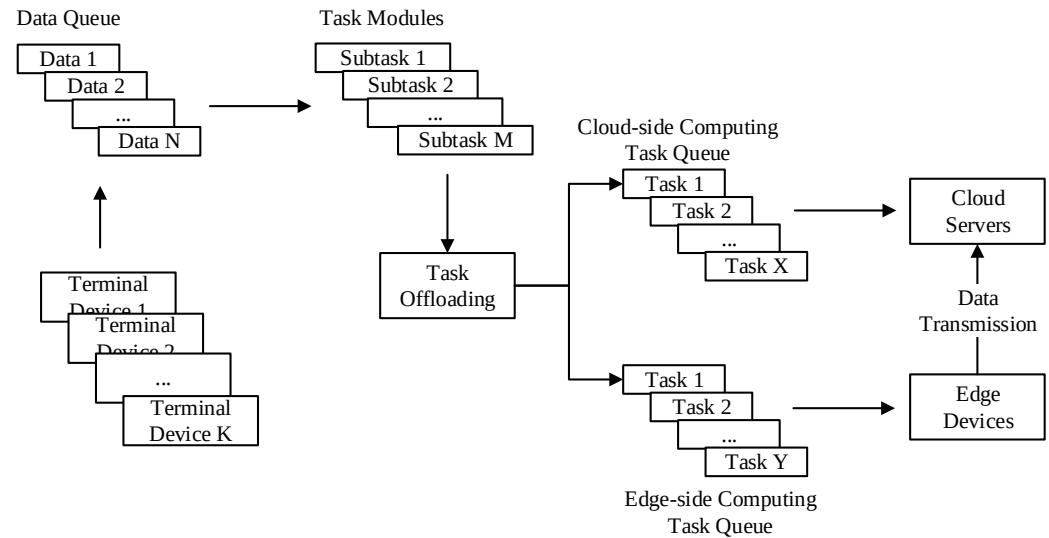


Figure 6. Task offloading.

3.2.3. Intelligence

Currently, industrial intelligence is a hot research topic. However, deploying artificial intelligence algorithms into practical scenarios still faces significant obstacles. Compared to executing algorithms in test environments and obtaining results, packaging algorithms into applications usable in IoT scenarios still requires considerable effort. Especially after a phase of system completion, integrating new algorithms often becomes difficult, making it hard for IIoT systems to keep up with the latest developments in the rapidly evolving field of artificial intelligence. In the previously mentioned five-layer architecture for IIoT, the platform layer actually takes on the responsibility of collecting data downwards and supporting applications upwards. Considering the demands of popular applications such as federated learning [52] for intelligent algorithm scheduling and operational support platforms, based on the platform layer's capabilities, it is possible to solve the data access issues from various sources and allow algorithm output results to directly control devices, thereby converging the problem of IIoT intelligence implementation to how to support algorithm development and operation on the platform layer. We believe that constructing management and support for algorithms in the IoT OS platform should be a key capability of the next generation of IIoT infrastructure.

To build support for algorithms on the IoT OS platform, we propose life cycle management for artificial intelligence algorithms applicable to the IoT OS platform, along with a unified algorithm interface standard. By implementing flexible, low-cost algorithm integration, it is possible to improve the current stagnation in intelligent construction of IIoT and apply excellent algorithms from various scenarios to practical applications.

In summary, the IoT OS platform adheres to the following principles, making the development, deployment, and use of algorithms in IIoT scenarios more flexible and effective.

Life cycle Management. We define the life cycle of an algorithm as the five stages shown in Figure 7: analysis, development, verification, operation, and maintenance. In the analysis stage, by visualizing and initially processing data collected in the IoT system, it is possible to assist in discovering the feasibility and design direction of algorithm construction, clarify the goals and scope of the algorithm, and further define the performance indicators of the algorithm. Since the IoT OS platform has the data collection capability mentioned earlier, it can use the collected data to choose an appropriate algorithm architecture to build a viable artificial intelligence model. Further, in the verification stage, by adjusting parameters and observing trial run results and fixing possible errors and issues, the development effect of the algorithm is confirmed to ensure normal operation under various conditions. In the operation stage, most current IoT platforms usually embed algorithms, only allowing core developers to adjust and modify algorithms. However, we

believe that the capability to deploy and run algorithms should be opened to users. By building a series of operational and maintenance feature sets, continuous evaluation and adjustment of the algorithm's performance are still possible after deployment. Constructing a complete life cycle process can improve the current lack of flexibility in existing IIoT applications for algorithms and ensure the high quality and reliability of algorithms.

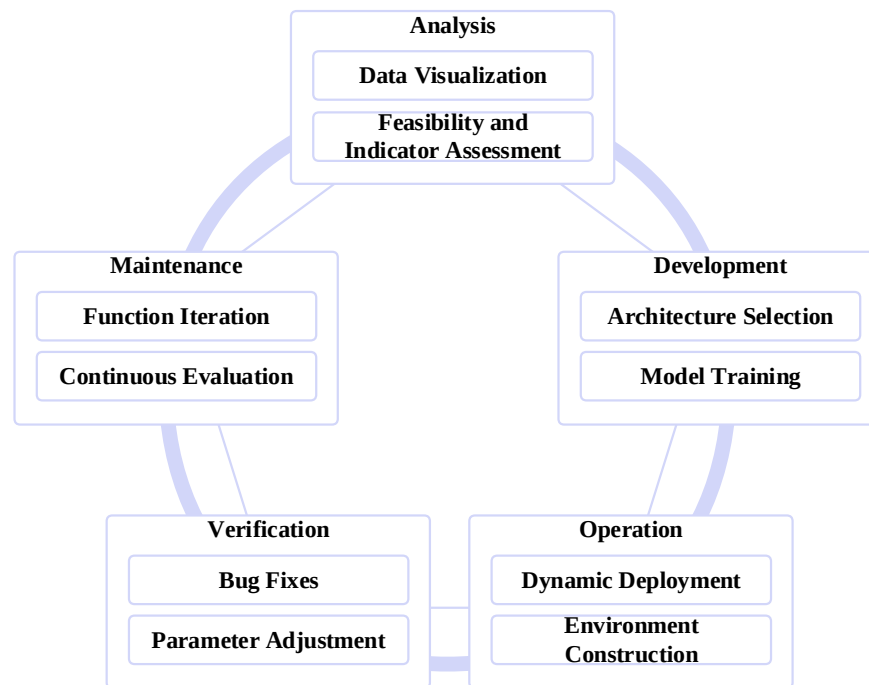


Figure 7. Life cycle of algorithms on the IoT OS platform.

Unified Interface Standard. In building the interface standard, we abstracted each algorithm as a stateful executor, inputting data in the required format of the algorithm and outputting usable inference results. Considering that most mainstream programming languages support dynamically loading external modules at runtime, we expected that for algorithms to be integrated, only an additional encapsulation layer would be needed to adapt to the given interface standard, allowing the system to load and unload modules at runtime. Therefore, the IoT operating system platform defines the interface primitives shown in Table 1 in as concise a manner as possible to support algorithm life cycle management.

Table 1. Unified interface definition.

Interface	Function
load	Load the algorithm into the system
unload	Unload the algorithm from the system
inference	Input data in the format required by the algorithm, perform inference, and return the results
info	Output the algorithm's self-descriptive information in a standard format
config	Set the related parameters of the algorithm

Traditional IoT platforms, such as AWS IoT and Alibaba Cloud's IoT platform, involve complex steps and processes for loading, reasoning, and unloading algorithm models. In contrast, the dynamic loading method we proposed focuses more on simplifying the process, emphasizing that by explicitly specifying the input and output formats and algorithm model files, the algorithm model can be easily loaded. Our design concept compares this

process to the system call of the operating system, making the use of the algorithm model more intuitive and efficient.

One of the main obstacles to applying algorithms at the development level is their closed nature, making maintenance during algorithm migration excessively challenging. To address this issue, we defined the info interface for obtaining algorithm runtime environment, performance requirements, input and output formats, and other information. By ensuring the self-descriptive capability of algorithms at the development end, application deployment usability is guaranteed.

3.2.4. Security

It should be recognized that integrating existing systems and devices is a common practice in achieving data-driven transformation in IIoT scenarios. However, when establishing connections with various systems, security considerations of the original systems, such as using fixed passwords or admin accounts, are often overlooked for efficiency and cost reasons. This increases the security risks in the process of increasingly complex business interactions and system connections, a problem difficult to solve with underlying security mechanisms such as network protocol encryption. This article does not discuss the application of traditional security technologies, but rather proposes some innovations from the overall perspective of constructing IIoT security scenarios.

The IoT OS platform is positioned to integrate connections with various systems and devices. Constructing mechanisms to ensure business security based on the IoT OS platform can avoid the aforementioned issues. Based on traditional operating system implementations of multi-domain isolation, as shown in Figure 8, each upper-layer virtual machine is completely isolated from others in terms of processors, memory, and storage, communicating through the kernel.

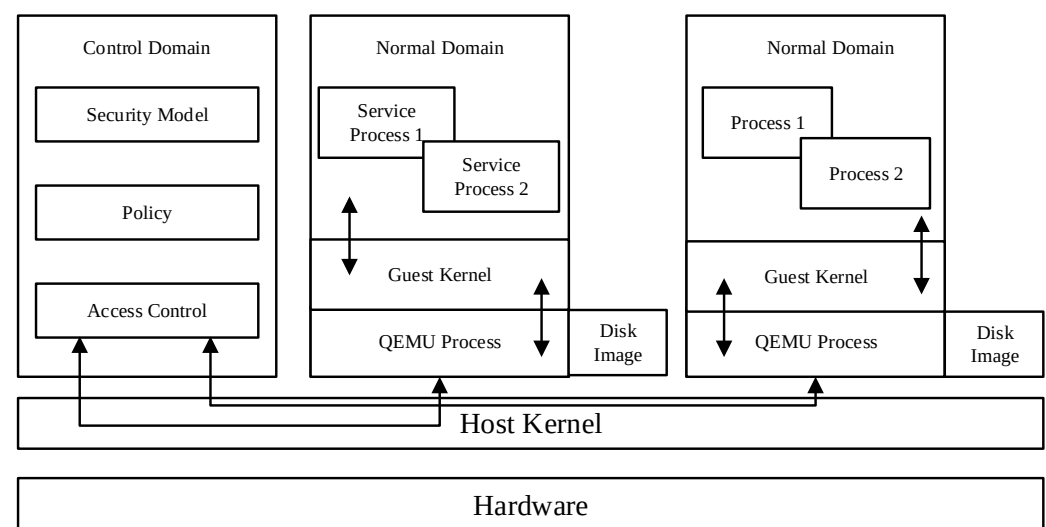


Figure 8. Multi-domain isolation of traditional OS.

On the IoT OS platform, we designed a multi-domain isolation model divided by business, including the following security rules:

- **Business Independence:** for a business constructed or integrated on the IoT OS platform, it is treated as an independent entity for management, with communication between businesses not arbitrarily allowed and only the minimal external interfaces opened.
- **Connection Ownership:** for individual permission accounts of accessed external systems or connections established with devices, the IoT operating system platform must attribute the permission of that connection to a specific business; it cannot be shared between businesses.
- **Data Isolation:** for data flowing through the IoT OS platform, it cannot be transmitted across businesses, with isolation implemented at both hardware and software levels.

- **Access Control:** establishing role-based access control (RBAC) and policy control, determining the operations and data access scope for each business to minimize potential security risks.

This business-divided multi-domain isolation model aims to build a framework that balances security, stability, and development efficiency. Considering the characteristic of existing digital transformations to integrate multiple existing systems and devices, viewing each business as an independent entity enables comprehensive isolation between businesses. This prevents potential threats and faults from spreading within a single business, thereby ensuring the security of the entire IIoT solution. By clearly attributing connection permissions to specific businesses, it facilitates responsibility tracing and system auditing, and prevents the misuse of connection permissions. Implementing data isolation, whether at the hardware or software level, helps prevent sensitive information leakage, ensuring data is not transmitted across businesses without authorization, thus enhancing privacy protection and the overall trustworthiness of the system. Moreover, the design of security mechanisms on the IoT operating system platform is independent of the underlying security technology stack, seamlessly integrating with security chips at the hardware level or encryption algorithms at the software level.

3.3. Comparison with Other IoT/IIoT Platforms

In this section, we compare the current mainstream IoT or IIoT platforms to measure the differences between IoT OS Platform and others in key indicators. We referred to the work of Hazra et al. [5] and investigated and tracked several IoT platforms that still performed well and continued to be developed and maintained in the following years. The comparison results are shown in Table 2.

Table 2. Comparison with other IoT platforms.

IoT Platform	Local Deployment	Real-Time Analysis	External Algorithm	Business Isolation	Custom Protocol Parsing	Modular Construction
Amazon AWS	×	✓	×	Easy	×	×
Microsoft Azure	✓	✓	×	Easy	×	×
GE Predix	×	✓	✓	Hard	×	×
SAP Leonardo	×	✓	×	Hard	×	×
Siemens Mindsphere	×	✓	✓	Hard	×	×
IoT OS Platform	✓	✓	✓	Easy	✓	✓

Through the above comparison, it can be observed that most IoT solutions currently focus on real-time construction, and therefore perform well. In terms of local deployment, there are few solutions that can support complete local deployment, as cloud based architectures typically provide more redundancy and stability. However, in some industrial scenarios, owners of industrial enterprises may be more resistant to cloud based service, fearing that it may lead to the leakage of their enterprise secrets. Therefore, they are more inclined to continue providing IoT products that are completely disconnected from external networks. The construction of external algorithm access capabilities is also an emerging direction in recent years, as discussed earlier, which can help users build more targeted intelligent functions. But currently, most IoT solutions have not considered this issue at the beginning of their design, making it difficult for them to follow up on subsequent construction. Business isolation can help ensure security at the business level. In most IoT solutions, this is usually achieved by introducing concepts such as "assets", but not all IoT solutions consider this. Custom protocol parsing and modular construction are unique features of the IoT OS Platform. As the definition of infrastructure, opening up the building

capabilities of the upper layer to users is believed to enhance the overall interoperability and usability of the solution.

3.4. Commonalities Between Classic OS and IoT OS Platform

In this section, we want to review classic OS architectures from a holistic perspective and explain how we measure the commonalities between classic OS and IoT OS Platform. The comparison between the two is shown in Figure 9.

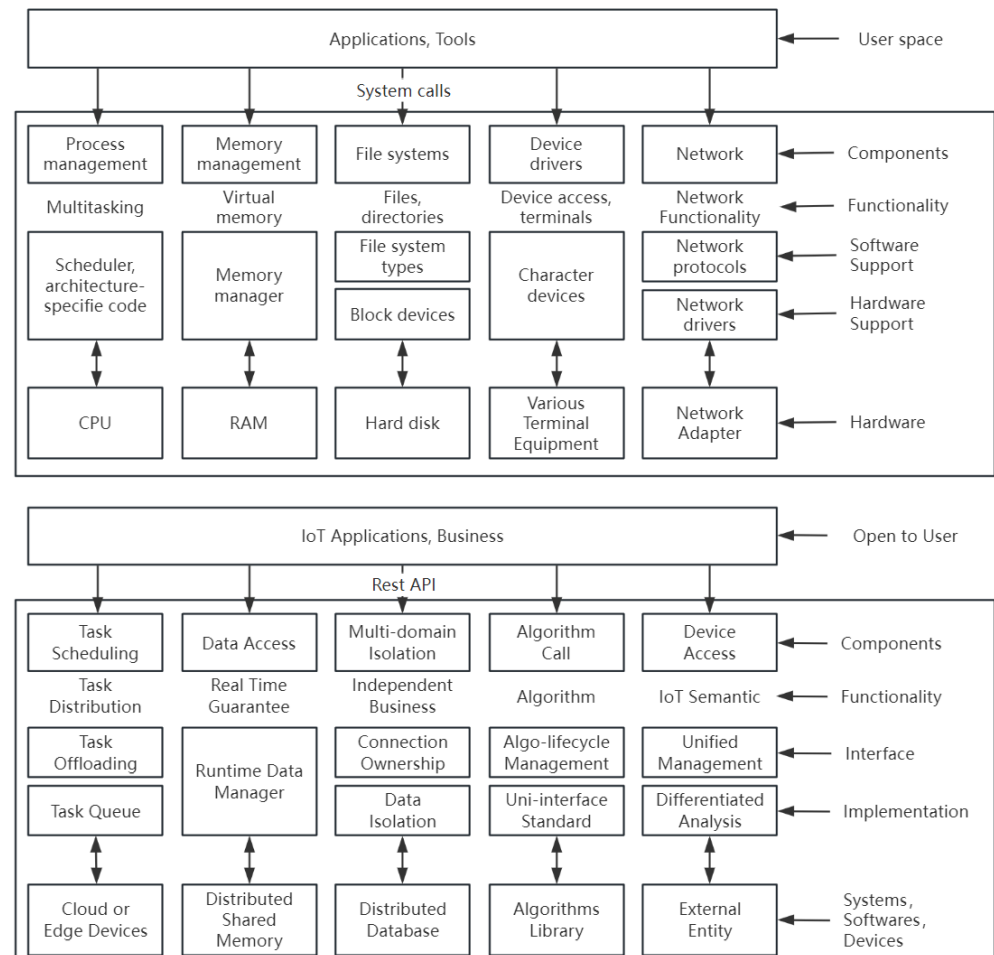


Figure 9. Commonalities between classic OS and IoT OS platform.

In a classic OS, the resources scheduled and managed include CPU, RAM, disks, external hardware devices, and network adapters. Among them, the CPU of the classic operating system can correspond to a cloud/edge computing device of an edge cloud collaborative solution, because in the Internet of Things scenario, these distributed devices provide the computing power required for business, and the classic operating system, based on CPU's upper layer process scheduling, can correspond to the task scheduling design we mentioned earlier.

The classic OS hosts the runtime data of the system to RAM, enabling fast real-time access. In the IoT OS Platform, we also built a nearby data read and write mechanism for stateful data during system operation, based on distributed shared memory components to support upper level business.

In terms of permission mechanisms, traditional operating systems typically utilize file systems to implement security policies by applying specific permissions (such as execute, read, and write) to files and handles. The multi-domain isolation mechanism involved

in IoT OS Platform is to define different connections and data as "files" with ownership characteristics, and to isolate and manage them.

Classic OS use different device drivers to operate different external hardware and typically support hot swapping of different hardware. In the next generation of industrial IoT, we believe that AI algorithms are the key external devices in IoT systems. Taking inspiration from classic OS, we have designed the invocation of algorithms and their lifecycle management to have a standard invocation interface and support for dynamic loading and unloading.

Considering the management of different network protocols in classic operating systems, we applied this idea to the practice of improving the interoperability of the Internet of Things. Based on different protocol stacks, we expect the IoT OS Platform to be able to access various external entities including systems, devices, and software by customizing the implementation of the upper layer.

4. Implementation Case Study

To verify the role of the IoT OS Platform in IIoT solutions, we constructed PFSP: IoT OS Platform For Smart Power Plants.

4.1. Problem Description

In a large-scale power plant, its intelligent transformation usually revolves around safety production. In this case, nearly 200 IPCs(IP Cameras) shown in Figure 10 were deployed in the power plant to provide video streams at various points, and we needed to build a platform on a server with limited hardware performance (shown in Table 3) to handle more than ten algorithms such as high-altitude operations, personnel falls, not wearing helmets on the video stream, and push alarms to users. The specific requirements are shown in Table 4. The system has significantly different characteristics from other past systems, with clear evaluation indicators for its accuracy and real-time performance of the algorithm proposed in the requirements, and the algorithm is required to be able to load/unload dynamically.

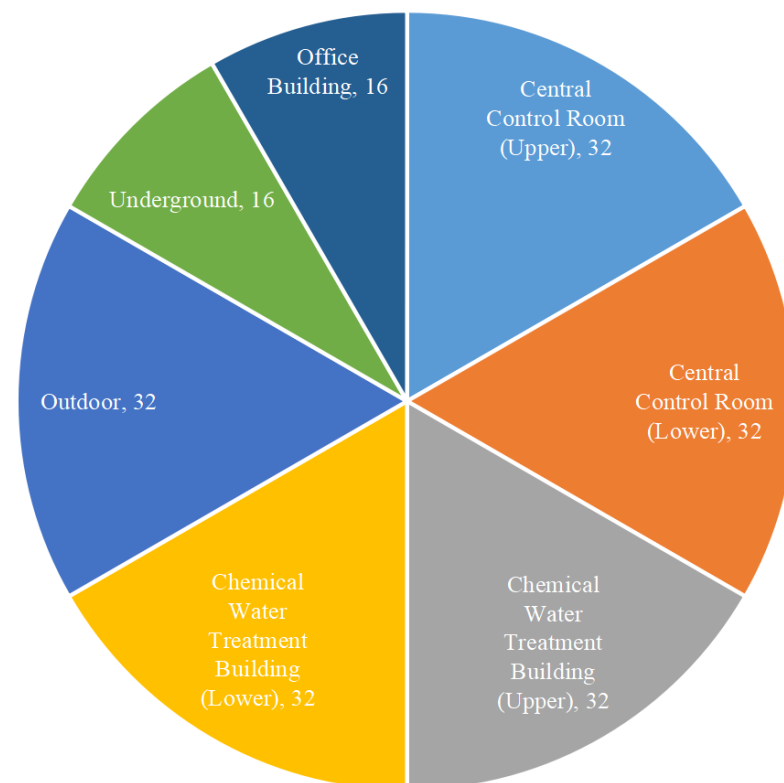


Figure 10. Number of cameras in each region.

Table 3. Server hardware configuration.

Item	Model	Specification
CPU	2 * Intel(R) Xeon(R) Silver 4310 CPU	2.10 GHz, 12 cores, 24t hreads
GPU	4 * NVIDIA GeForce RTX 3080	10 GB Memory, 1440 MHz
Memory	6 * SKhynix HMA82GR7CJR4N-WM	16 GB, 2666 MT/s

Table 4. Algorithms' requirements.

Algorithm Name	Requirement Description
Working at Heights	Monitor the compliance of personnel wearing safety equipment in the screen, and determine whether protective railings are installed in high-risk areas for high-altitude operations
Hot Work	Detect and analyze whether the workers in the screen are performing hot work correctly, and determine whether one person is operating and one person is monitoring according to regulations
Lifting Operations	Determine whether there is anyone standing under the lifting arm in violation of regulations while the lifting equipment is performing lifting operations in the screen
Loitering Detection	Automatically identify personnel entering and exiting the monitoring area and continuously track the duration of stay for statistical purposes
Fatigue Work	Automatically identify whether there are workers who have been in the camera for a long time within the monitoring range
Digital Meter Reading	Identify the instrument readings in the video
Personnel Fall	Check if there is any falling behavior in the screen
Wearing of Safe Helmet	Testing the behavior of personnel wearing standard safety helmets correctly on their heads
Wearing Long Sleeves	Detect the positions of personnel and local target areas in the image, and judge the wearing of short sleeves based on compliance criteria
Using Mobile Phones	Identify the behavior of workers using mobile phones within the designated area
Smoking Detection	Real time monitoring of key fire prevention areas and designated monitoring and analysis points, and detection of smoking violations by power station personnel
Sleeping on Duty	Automatically identify whether personnel are sleeping or leaving their posts within the monitoring range
Cross Border Invasion	Automatically identify whether personnel have entered hazardous areas or entered designated areas in violation of regulations within the monitoring range
Facial Attendance	Analyze the real-time video stream obtained by the surveillance camera, automatically identify and compare whether it is an internal employee, and record it
Personnel Flow Count	Automatically identify the number of personnel within the monitoring range
Flames Identification	Identify the presence of flames in the video

In addition, we also needed to access IoT devices including access gates, environmental monitoring systems, etc., to comprehensively display various types of data on the platform.

The overall connection diagram architecture is shown in Figure 11. The cameras in each region were first connected to the NVR (Network Video Recorder) deployed in the same area through corresponding switches, and then the NVR aggregated video data to the central switch. The central switch was connected to the main server through LAN 1 port, allowing the central server to access and control the cameras. In addition, access control devices and meteorological monitoring servers also established a connection with the main server to transmit specific IoT devices to the platform. In order to speed up reasoning and reduce transmission bandwidth pressure, we deployed edge computing devices in the corresponding regions to complete business reasoning in the regions. These edge devices also established communication with the main server based on the aforementioned network connection architecture, in order to receive control commands from the main server. The

main server had two network ports, with two LAN ports connected to the internal network of the office area. Users were able to access the platform through this internal network.

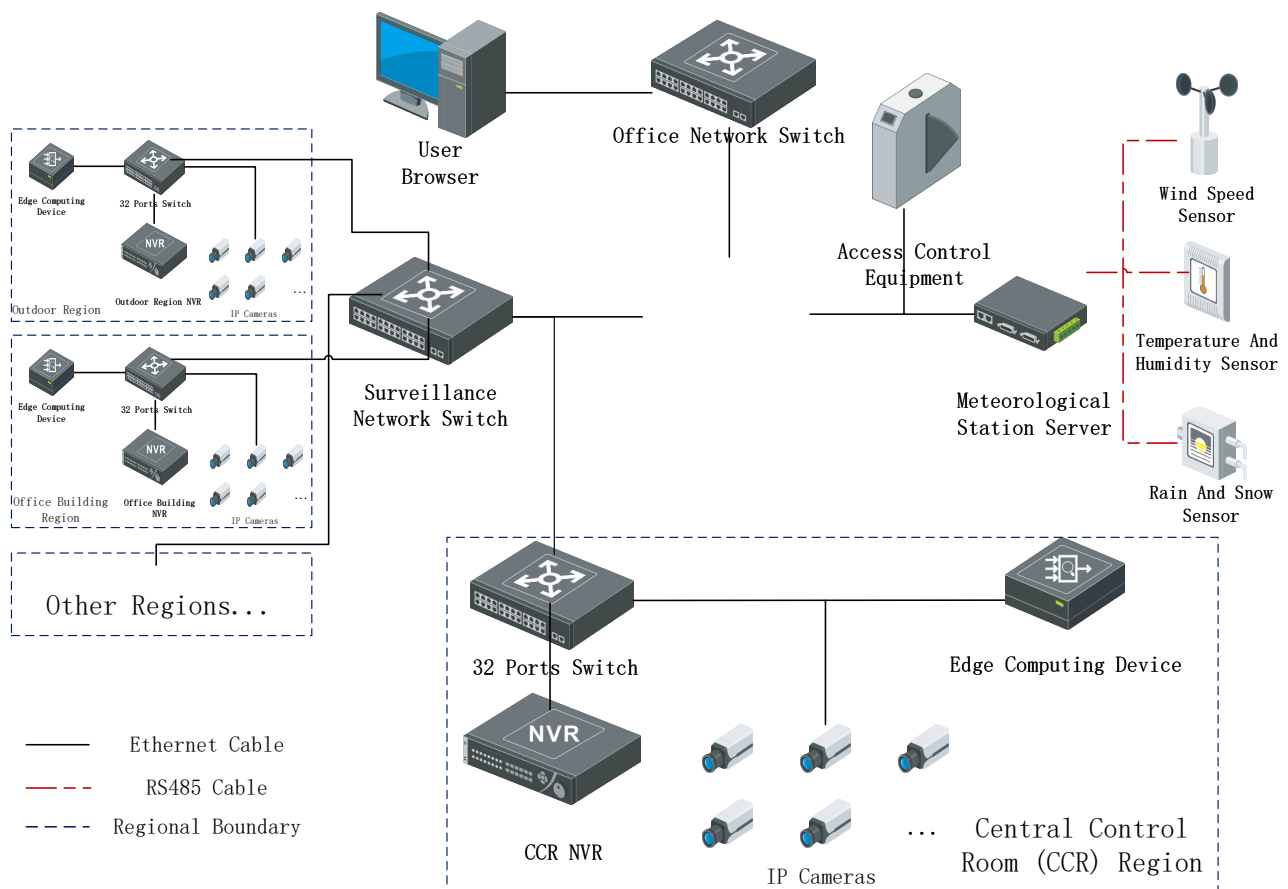


Figure 11. Overview topology diagram of the device connections. Each area includes a regional switch, an NVR, at least one edge computing device, and many IPCs. IoT OS Platform is deployed on the Main Server, which has dual network ports connecting to both the office network and the surveillance network, allowing users in the office network to access it normally.

4.2. Implementation

Based on the design concept of the IoT OS Platform we discussed earlier, we have adopted the following solutions to address the difficulties in this project.

4.2.1. Heterogeneous Device Access

In the construction process of this system, we needed to connect the cameras and IoT sensors of various protocol types. The specific devices and their protocol types are shown in Table 5.

Table 5. Devices and protocols.

Device Type	Protocol
Camera	Private Protocol Based on TCP
Access Equipment	Private Protocol Based on TCP
Meteorological Monitoring System	JSON on HTTP
Leakage Monitoring System	JSON on HTTP

Considering the urgency of project construction, in order to be able to access these devices in a short period of time, we applied the design concept of the access section

mentioned earlier. Various types of devices were abstracted into unified entities for integration, and corresponding semantic standards were constructed to ensure overall system interoperability on the basic software architecture.

Taking the interface of the leakage monitoring system as an example, we extracted and mapped the original interface information, as shown in Table 6.

Table 6. Mapping raw API to semantic.

Raw API	Semantic in IoT OS Platform
POST /api/v1/device/getStatus	read
POST /api/v1/device/setStatus	write
POST /api/v1/login	open
POST /api/v1/logout	close
POST /api/v1/alarm/getAlarmLogList	exception

When accessing the access gates, as it does not have an open HTTP-based network interface, we needed to map its call method to the general semantics of the IoT OS Platform, as shown in Table 7.

Table 7. Mapping call methods to semantic.

Raw API	Semantic in IoT OS Platform
boolean CLIENT_QueryDevState	read
boolean CLIENT_ControlDeviceE	write
LLong CLIENT_LoginWithHighLevelSecurity	open
boolean CLIENT_Logout	close
interface fMessCallBack	exception

4.2.2. Real Time Alarm Guarantee

As described in the previous section, compared to other systems in the past, we needed to perform strong real-time identification and alarm within a large factory area. Combined with our previous design ideas for data access and task scheduling in the IoT OS Platform, we introduced several edge computing devices. According to the geographical location of the cameras, we sent the video stream address and its inference tasks to the nearby edge nodes for calculation, and then collected the alarm data to the central platform for display.

After completing deployment and scheduling, its data flow architecture is shown in Figure 12.

During the process, we mainly focused on how to reduce the transmission time spent on business. According to the timing chart, our timing rules are shown in Figure 13. Firstly, the camera transmits the RTSP encoded stream data to the frame processing process on the edge device. After the frame data is extracted, we record this timestamp globally. Then, the frame processing process will place this frame data into a shared memory for the inference process to use. After the inference process reads this frame of data from the shared memory, we record the timestamp of the start of inference globally. After the inference is completed, the inference time can be obtained. Afterwards, the inference process will send the results to the central server, which will then present the results to the browser. On the central server, when data are pushed to the browser, we record the timestamp of the end of transmission to obtain the total time spent. Afterwards, we calculate the transmission time by subtracting the two. In order to minimize the impact of random disturbances as much as possible, we conducted 100 tests on the workflow of each algorithm and took its average value.

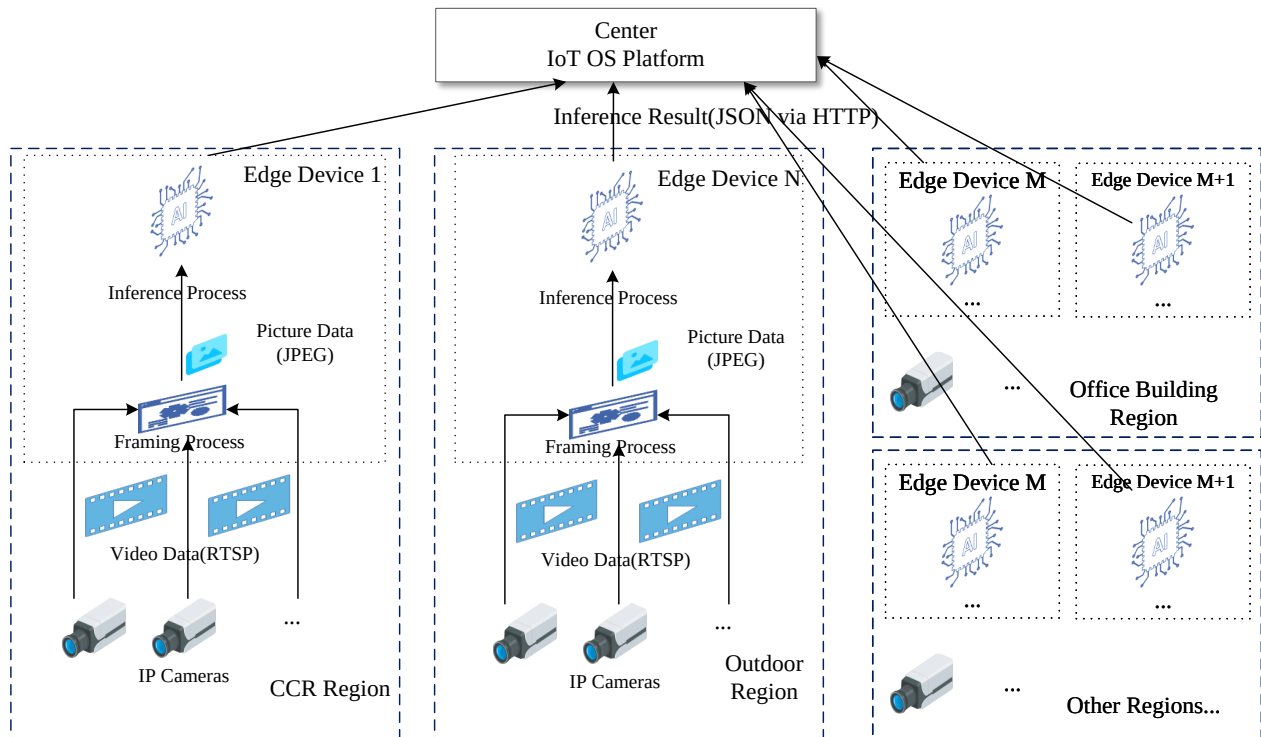


Figure 12. The inference data flow architecture.

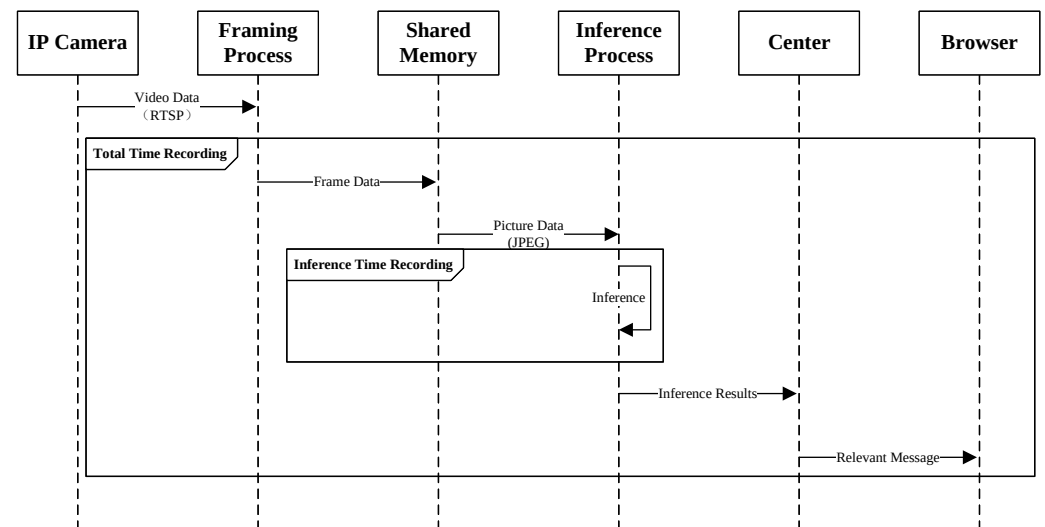


Figure 13. Time consumption calculation method.

We conducted performance tests on 16 algorithms deployed in the system, and counted the time it took for these algorithms to receive alarm results from accessing video streams to the browser end, as shown in Table 8.

It can be observed that in the system, the total time for identifying anomalies and issuing alarms is basically less than 0.2 s. Among them, the Fatigue Work and the Sleep on Duty algorithm have the least total time consumption, and the YOLOv8 algorithm is used at the bottom of both, so they have good performance in inference. The personnel fall algorithm uses an ONNX format model for posture recognition, but its optimization is relatively poor, resulting in higher inference time. Although the inference time is algorithm related and fluctuates greatly, the calculated transmission delay remains within a stable range of 70–84 milliseconds. By calculating the proportion of transmission time to total process time, it is possible to better observe and confirm that transmission will not become a bottleneck in the real-time performance of the system. In summary, by constructing a

complete workflow of frame retrieval inference alarm, it can be verified that the IoT OS Platform performs well in real-time design.

Table 8. Time cost of system.

Algorithm Name	Inference Cost (ms)	Transmission Cost (ms)	Total Cost (ms)	Transmission Cost Percentage (%)
Working at Heights	97.35	74.23	171.58	43.26
Hot Work	98.57	70.75	169.50	41.74
Lifting Operations	98.05	72.15	170.20	42.39
Loitering Detection	65.53	73.43	138.96	52.84
Fatigue Work	36.68	77.21	113.89	67.79
Personnel Fall	121.04	84.00	205.04	40.97
Wearing of Safe Helmet	71.79	72.45	144.24	50.23
Wearing Long Sleeves	97.08	78.22	175.30	44.62
Using Mobile Phones	67.19	80.82	148.01	54.60
Smoking Detection	96.69	80.92	177.61	45.56
Sleeping on Duty	35.86	76.81	112.67	68.17
Cross Border Invasion	70.97	75.36	141.33	49.78
Facial Attendance	106.83	72.36	179.19	40.38
Personnel Flow Count	70.15	70.64	140.79	50.17
Flames Identification	66.90	70.85	137.75	51.43

4.2.3. Algorithm Management

In the requirements, it is clearly stated that there is a need for dynamic loading and unloading of algorithms, which coincides with our view on the industrial Internet of Things mentioned earlier. Therefore, we have designed a management function for the algorithm in the system according to the five stages of the algorithm lifecycle mentioned earlier:

Development: In this case, since almost all requirements are object detection, we choose PyTorch and YOLO [53] framework to develop the algorithms.

Operation: We have developed a user interface that offers interactive capabilities, allowing users to visually upload and modify algorithms and configure related settings, such as detection thresholds and areas, as shown in Figures 14 and 15.

Verification and Maintenance: Using the user interface mentioned above, we can deploy any algorithm dynamically and easily. Also, we use container to provide running environment, omitting the process of environment configuration. With these two methods, it is possible to achieve algorithm iteration and upgrades.

Analysis: We have developed a false positive management system, which facilitates the manual labeling of misclassified images. This feature aids in the collection of false positive data, enabling the refinement of algorithms that are currently in deployment.

In addition, during the construction process, we often encounter various algorithm recognition requirements with unique business characteristics, such as unsafe behaviors such as smoking and not wearing helmets in monitoring scenarios. In such special needs, using naive object detection algorithms can easily lead to frequent recognition errors. Taking smoking behavior as an example, detecting only cigarette can easily generate false positives as shown in Figure 16.

Figure 14. Algorithm management function. It enables the integration of various algorithms within the platform, significantly simplifying the management costs associated with multiple algorithms.

Figure 15. Algorithm settings. We have extracted the common key configurations of the algorithms. Through these configurations, it is possible to dynamically adjust various parameters of the algorithms based on their requirements in different scenarios.



Figure 16. False positive of smoking. These four images are all from the Central Control Room (Upper). Due to various reasons, some types of images may be misidentified. In particular, the image in the bottom right corner could also be mistaken for smoking by human eyes.

To address this issue, we leveraged the design advantages of IoT OS Platform by sending single-frame data to multiple algorithm execution entities for parallel processing, and then summarized the results. Taking smoking behavior as an example, we sent the same frame image to two algorithm execution bodies for personnel detection and cigarette detection, and then took the intersection of the recognition of the head and the recognition of cigarettes in the results. The process is shown in Figure 17. By adopting this approach, we reduced the false alarm rate by more than half.

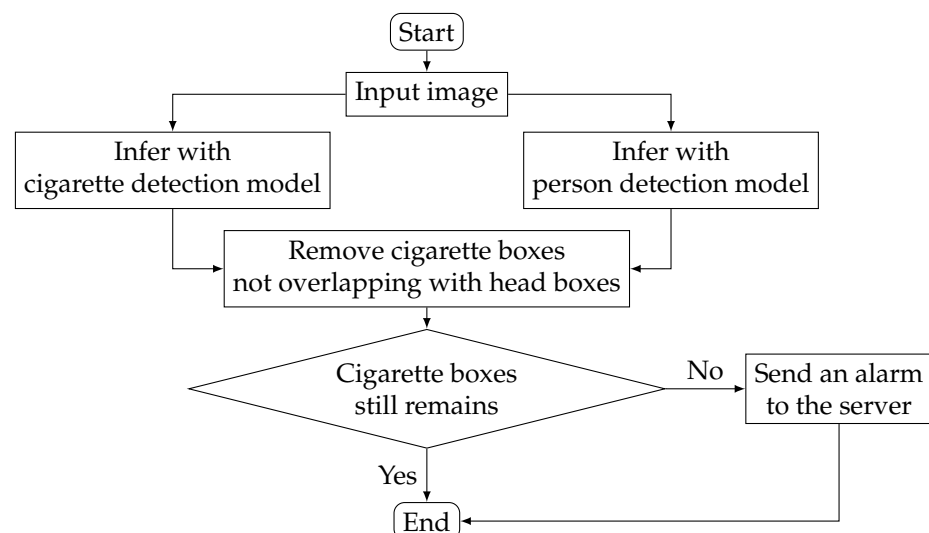


Figure 17. Parallel processing for smoking detection. Using multiple algorithms in parallel for inference, as opposed to relying on a single model, can yield more accurate detection results.

5. Conclusions and Outlook

Currently, the industrial internet is at the beginning stage of practice, while in the vertical industries of the industrial internet, it is at the stage of having essentially completed digitalization and partially achieved intelligence. As the underlying support for the industrial internet, IIoT systems directly interface with onsite equipment, databases, information management systems, and other entities. Their capabilities directly impact the breadth and depth of intelligence in upper-layer industrial applications. Without the support of lower-layer IoT system capabilities, the advancement of industrial intelligence will be extremely difficult.

The IoT OS platform, based on standardized, systematic, and modular design philosophies and combined with specific business scenarios, enables the customized development of industry-specific IoT systems on a reliable, generic framework. This holds instructive significance for the construction of IoT systems in industrial IoT scenarios. Against the backdrop of the traditional internet application domain approaching saturation, building intelligent applications for the industrial sector will become a new blue ocean. As the industrial internet continues to evolve, IoT operating system platforms will continue to play a key role, driving innovation and development in intelligent manufacturing, industrial automation, and other fields.

Although IoT OS platforms can already solve the most critical issues in current IIoT scenarios, we believe that the future ecosystem of IIoT will transition from closed to open. This transition creates a demand for openness in IoT OS platforms. To address this demand, it could be considered to provide channels for expanding applications, offer incentives for developers to contribute, attract component and application developers, thereby fostering the formation of an ecosystem around the platform from the bottom up.

In future developments, IoT OS platforms need to continually adapt to the evolution of new technologies and market demand changes, including challenges in edge computing, artificial intelligence, security, and privacy protection. By overcoming these challenges, IoT operating system platforms will provide a more powerful, flexible, and secure software infrastructure for the industrial IoT domain, thus promoting the continuous prosperity and development of the entire IoT ecosystem.

Author Contributions: Conceptualization, Z.X., Y.L., Z.S. and D.Y.; methodology, Z.X., Z.S., H.Z. and Y.Y.; project administration & supervision, Z.X., Y.L. and D.Y.; software, Z.X., Z.S. and Y.Y.; writing—original draft, Z.X., Z.S. and H.Z.; writing—review & editing, Z.X., Z.S., Y.L., D.Y. and X.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Due to concerns regarding industrial production safety and confidentiality agreements, the specific data used in case study are not publicly available.

Conflicts of Interest: Authors Zhongjie Sun, Han Zheng and Dan Yu were employed by the company China Standard Intelligent Security 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.

References

1. Ashton, K. That ‘Internet of Things’ thing. *RFID J.* **2009**, *22*, 97–114.
2. Strategy, I.; Unit, P. *The Internet of Things*; International Telecommunication Union: Geneva, Switzerland, 2005.
3. Mineraud, J.; Mazhelis, O.; Su, X.; Tarkoma, S. A gap analysis of Internet-of-Things platforms. *Comput. Commun.* **2016**, *89*, 5–16. [[CrossRef](#)]
4. Liu, Y.; Yang, C.; Jiang, L.; Xie, S.; Zhang, Y. Intelligent edge computing for IoT-based energy management in smart cities. *IEEE Netw.* **2019**, *33*, 111–117. [[CrossRef](#)]

5. 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. [\[CrossRef\]](#)
6. De Vita, F.; Bruneo, D.; Das, S.K. A novel data collection framework for telemetry and anomaly detection in industrial iot systems. In Proceedings of the 2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI), Sydney, Australia, 21–24 April 2020; pp. 245–251.
7. Terroso-Saenz, F.; González-Vidal, A.; Ramallo-González, A.P.; Skarmeta, A.F. An open IoT platform for the management and analysis of energy data. *Future Gener. Comput. Syst.* **2019**, *92*, 1066–1079. [\[CrossRef\]](#)
8. Zuehlke, D. SmartFactory—Towards a factory-of-things. *Annu. Rev. Control.* **2010**, *34*, 129–138. [\[CrossRef\]](#)
9. Hejazi, H.; Rajab, H.; Cinkler, T.; Lengyel, L. Survey of platforms for massive IoT. In Proceedings of the 2018 IEEE International Conference on Future IoT Technologies (Future IoT), Eger, Hungary, 18–19 January 2018; pp. 1–8.
10. Chowdhury, A.; Raut, A.S. Benefits, challenges, and opportunities in adoption of industrial IoT. *Int. J. Comput. Intell. Iot* **2019**, *2*, 4.
11. Malik, P.K.; Sharma, R.; Singh, R.; Gehlot, A.; Satapathy, S.C.; Alnumay, W.S.; Pelusi, D.; Ghosh, U.; Nayak, J. Industrial Internet of Things and its applications in industry 4.0: State of the art. *Comput. Commun.* **2021**, *166*, 125–139. [\[CrossRef\]](#)
12. Weyrich, M.; Ebert, C. Reference architectures for the internet of things. *IEEE Softw.* **2015**, *33*, 112–116. [\[CrossRef\]](#)
13. Mashal, I.; Alsaryrah, O.; Chung, T.Y.; Yang, C.Z.; Kuo, W.H.; Agrawal, D.P. Choices for interaction with things on Internet and underlying issues. *Ad Hoc Netw.* **2015**, *28*, 68–90. [\[CrossRef\]](#)
14. Said, O.; Masud, M. Towards internet of things: Survey and future vision. *Int. J. Comput. Netw.* **2013**, *5*, 1–17.
15. Wu, M.; Lu, T.J.; Ling, F.Y.; Sun, J.; Du, H.Y. Research on the architecture of Internet of Things. In Proceedings of the 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE), Chengdu, China, 20–22 August 2010; Volume 5, pp. V5-484–V5-487.
16. Sethi, P.; Sarangi, S.R. Internet of things: architectures, protocols, and applications. *J. Electr. Comput. Eng.* **2017**, *2017*, 9324035. [\[CrossRef\]](#)
17. Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Commun. Surv. Tutorials* **2015**, *17*, 2347–2376. [\[CrossRef\]](#)
18. Tan, L.; Wang, N. Future internet: The internet of things. In Proceedings of the 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE), Chengdu, China, 20–22 August 2010; Volume 5, pp. V5-376–V5-380.
19. Khan, R.; Khan, S.U.; Zaheer, R.; Khan, S. Future internet: the internet of things architecture, possible applications and key challenges. In Proceedings of the 10th International Conference on Frontiers of Information Technology, Islamabad, Pakistan, 17–19 December 2012; pp. 257–260.
20. Da Xu, L.; He, W.; Li, S. Internet of things in industries: A survey. *IEEE Trans. Ind. Inform.* **2014**, *10*, 2233–2243.
21. Shu, Z.; Wan, J.; Zhang, D.; Li, D. Cloud-integrated cyber-physical systems for complex industrial applications. *Mob. Netw. Appl.* **2016**, *21*, 865–878. [\[CrossRef\]](#)
22. Bonomi, F.; Milito, R.; Natarajan, P.; Zhu, J. Fog computing: A platform for internet of things and analytics. In *Big Data and Internet of Things: A Roadmap for Smart Environments*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 169–186.
23. Aazam, M.; Zeadally, S.; Harras, K.A. Deploying fog computing in industrial internet of things and industry 4.0. *IEEE Trans. Ind. Inform.* **2018**, *14*, 4674–4682. [\[CrossRef\]](#)
24. Sisinni, E.; Saifullah, A.; Han, S.; Jennehag, U.; Gidlund, M. Industrial internet of things: Challenges, opportunities, and directions. *IEEE Trans. Ind. Inform.* **2018**, *14*, 4724–4734. [\[CrossRef\]](#)
25. Atzori, L.; Iera, A.; Morabito, G. Siot: Giving a social structure to the internet of things. *IEEE Commun. Lett.* **2011**, *15*, 1193–1195. [\[CrossRef\]](#)
26. Åkerberg, J.; Gidlund, M.; Björkman, M. Future research challenges in wireless sensor and actuator networks targeting industrial automation. In Proceedings of the 9th IEEE International Conference on Industrial Informatics, Lisbon, Portugal, 26–29 July 2011; pp. 410–415.
27. Hankel, M.; Rexroth, B. The reference architectural model industrie 4.0 (rami 4.0). *Zvei* **2015**, *2*, 4–9.
28. Flatt, H.; Schriegel, S.; Jasperneite, J.; Trsek, H.; Adamczyk, H. Analysis of the Cyber-Security of industry 4.0 technologies based on RAMI 4.0 and identification of requirements. In Proceedings of the 21st IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Berlin, Germany, 6–9 September 2016; pp. 1–4.
29. Lin, S.W.; Miller, B.; Durand, J.; Joshi, R.; Didier, P.; Chigani, A.; Torenbeek, R.; Duggal, D.; Martin, R.; Bleakley, G.; et al. *Industrial Internet Reference Architecture*; Technical Report; Industrial Internet Consortium (IIC): Boston, MA, USA, 2015.
30. Alliance, O.M. *Lightweight Machine to Machine Technical Specification*; Open Mobile Alliance: San Diego, CA, USA, 2017; Volume 1.
31. Morato, A.; Vitturi, S.; Tramarin, F.; Cenedese, A. Assessment of different OPC UA implementations for industrial IoT-based measurement applications. *IEEE Trans. Instrum. Meas.* **2020**, *70*, 5500911. [\[CrossRef\]](#)
32. Karaagac, A.; Verbeeck, N.; Hoebeke, J. The integration of LwM2M and OPC UA: An interoperability approach for industrial IoT. In Proceedings of the 5th IEEE World Forum on Internet of Things (WF-IoT), Limerick, Ireland, 15–18 April 2019; pp. 313–318.
33. Centenaro, M.; Costa, C.E.; Granelli, F.; Sacchi, C.; Vangelista, L. A survey on technologies, standards and open challenges in satellite IoT. *IEEE Commun. Surv. Tutorials* **2021**, *23*, 1693–1720. [\[CrossRef\]](#)
34. Wang, F.; Zhang, M.; Wang, X.; Ma, X.; Liu, J. Deep learning for edge computing applications: A state-of-the-art survey. *IEEE Access* **2020**, *8*, 58322–58336. [\[CrossRef\]](#)

35. Porambage, P.; Okwuibe, J.; Liyanage, M.; Ylianttila, M.; Taleb, T. Survey on multi-access edge computing for internet of things realization. *IEEE Commun. Surv. Tutorials* **2018**, *20*, 2961–2991. [\[CrossRef\]](#)
36. Walia, G.K.; Kumar, M.; Gill, S.S. Ai-empowered fog/edge resource management for iot applications: A comprehensive review, research challenges and future perspectives. *IEEE Commun. Surv. Tutorials* **2023**, *26*, 619–669. [\[CrossRef\]](#)
37. Ding, X.; Wang, H.; Li, G.; Li, H.; Li, Y.; Liu, Y. IoT data cleaning techniques: A survey. *Intell. Conver. Netw.* **2022**, *3*, 325–339. [\[CrossRef\]](#)
38. Yang, J.C.; Lai, C.L.; Sheu, H.T.; Chen, J.J. An intelligent automated door control system based on a smart camera. *Sensors* **2013**, *13*, 5923–5936. [\[CrossRef\]](#) [\[PubMed\]](#)
39. Althubiti, S.A.; Alenezi, F.; Shitharth, S.; Reddy, C.V.S. Circuit Manufacturing Defect Detection Using VGG16 Convolutional Neural Networks. *Wirel. Commun. Mob. Comput.* **2022**, *2022*, 1070405. [\[CrossRef\]](#)
40. Wu, P.; Lu, Z.; Zhou, Q.; Lei, Z.; Li, X.; Qiu, M.; Hung, P.C. Bigdata logs analysis based on seq2seq networks for cognitive Internet of Things. *Future Gener. Comput. Syst.* **2019**, *90*, 477–488. [\[CrossRef\]](#)
41. Cook, A.A.; Mısırlı, G.; Fan, Z. Anomaly detection for IoT time-series data: A survey. *IEEE Internet Things J.* **2019**, *7*, 6481–6494. [\[CrossRef\]](#)
42. Bouarourou, S.; Zannou, A.; Boulaalam, A. IoT Based Smart Agriculture Monitoring System with Predictive Analysis. In Proceedings of the 2nd International Conference on Innovative Research in Applied Science, Engineering and Technology (IRASET), Meknes, Morocco, 3–4 March 2022; pp. 1–5.
43. Bodkhe, U.; Tanwar, S.; Parekh, K.; Khanpara, P.; Tyagi, S.; Kumar, N.; Alazab, M. Blockchain for industry 4.0: A comprehensive review. *IEEE Access* **2020**, *8*, 79764–79800. [\[CrossRef\]](#)
44. Dai, H.N.; Zheng, Z.; Zhang, Y. Blockchain for Internet of Things: A survey. *IEEE Internet Things J.* **2019**, *6*, 8076–8094. [\[CrossRef\]](#)
45. Schneider, S. The industrial internet of things (iiot) applications and taxonomy. In *Internet of Things and Data Analytics Handbook*; Wiley: Hoboken, NJ, USA, 2017; pp. 41–81.
46. Moura, R.; Ceotto, L.; Gonzalez, A.; Toledo, R. Industrial Internet of Things (IIoT) platforms-an evaluation model. In Proceedings of the 2018 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 12–14 December 2018; pp. 1002–1009.
47. Khin, S.; Kee, D.M.H. Factors influencing Industry 4.0 adoption. *J. Manuf. Technol. Manag.* **2022**, *33*, 448–467. [\[CrossRef\]](#)
48. Ukeh, M.G.; Shade, O.K.; Ogunojemite, Y.; Akinlade, B. An Overview of the Achievements of Operating System Architectures. *Int. J. Adv. Res. Comput. Sci.* **2014**, *5*, 266–269.
49. Jayasankar, U.; Thirumal, V.; Ponnurangam, D. A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications. *J. King Saud Univ. Comput. Inf. Sci.* **2021**, *33*, 119–140. [\[CrossRef\]](#)
50. Ganzha, M.; Paprzycki, M.; Pawłowski, W.; Szmaja, P.; Wasielewska, K. Semantic interoperability in the Internet of Things: An overview from the INTER-IoT perspective. *J. Netw. Comput. Appl.* **2017**, *81*, 111–124. [\[CrossRef\]](#)
51. Saadoon, M.; Hamid, S.H.A.; Sofian, H.; Altarturi, H.H.; Azizul, Z.H.; Nasuha, N. Fault tolerance in big data storage and processing systems: A review on challenges and solutions. *Ain Shams Eng. J.* **2022**, *13*, 101538. [\[CrossRef\]](#)
52. Khan, L.U.; Saad, W.; Han, Z.; Hossain, E.; Hong, C.S. Federated learning for internet of things: Recent advances, taxonomy, and open challenges. *IEEE Commun. Surv. Tutorials* **2021**, *23*, 1759–1799. [\[CrossRef\]](#)
53. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.

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.