AgriEngineering

MDPI

Article

A Hybrid Multi-Agent Control Architecture for Interoperable and Deterministic IoT-Based Swine Precision Feeding

Vicente López-Sacanell [1,2] and Lluís Miquel Plà-Aragonés [2,3,*]

[1] Research Group in AgroICT & Precision Agriculture (GRAP), Department of Agricultural and Forest Sciences and Engineering (DCEFA), Universitat de Lleida, 25198 Lleida, Spain; vicente.lopez@udl.cat
[2] Agrotecnio-CERCA Center, 25198 Lleida, Spain
[3] Department of Mathematics, Universitat de Lleida, 25001 Lleida, Spain
* Correspondence: lluismiquel.pla@udl.cat; Tel.: +34-973703318

Abstract

Precision Livestock Farming (PLF) requires real-time control systems that connect high-level Decision Support Systems with resource-constrained edge devices. This paper presents a hybrid Multi-Agent System (MAS) architecture for swine precision feeding designed to address the trade-off between semantic interoperability and real-time operational efficiency. The proposed Controlling Module uses a dual-layer communication strategy: a lightweight character-delimited TCP/IP protocol ensures deterministic performance for embedded controllers, while an XML-serialized format that maps to the FIPA Agent Communication Language preserves semantic interoperability. A custom serialization/deserialization algorithm was developed to process this XML structure within LabVIEW while avoiding the overhead typically associated with generic DOM/SAX parsers. The architecture was validated in a 120 h laboratory test that combined a Digital Twin simulation of 50 virtual feeders with Hardware-in-the-Loop testing of key sensing components. Under these test conditions, no communication failures were observed, all simulated network interruptions were recovered from, and the system operated with a modest resource footprint, including an average CPU use of 15% and a peak memory use of 350 MB. The platform also processed 2590 consumption events without reported data loss during the validation period. These results indicate that the proposed hybrid MAS architecture is a feasible solution for integrating interoperable decision support and deterministic edge control in PLF applications.

Keywords: precision livestock farming; precision feeding; multi-agent system; agricultural IoT; communication protocol; embedded control

Check for updates

Academic Editor: Lilong Chai

Received: 20 March 2026
Revised: 6 June 2026
Accepted: 10 June 2026
Published: 13 June 2026

Copyright: © 2026 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.

1. Introduction

Precision Livestock Farming (PLF) relies on digital systems capable of monitoring animals, processing heterogeneous data streams, and supporting management decisions in real time [1–4]. In swine production, precision feeding has attracted particular interest because feed represents a major share of total production costs and because individualized feeding strategies can reduce nutrient oversupply and associated nitrogen and phosphorus emissions [5–8]. However, the practical implementation of precision feeding depends not only on nutritional models but also on control architectures that are able to connect Decision Support Systems (DSS), sensors, and Automatic Feeding Systems (AFS) in a reliable and timely manner [9–12].

Agricultural environments make this integration difficult [13,14]. Unlike many industrial settings, farm systems combine heterogeneous hardware, intermittent connectivity, and devices with limited computational resources. In this context, deterministic operation becomes important because communication delays or inconsistent message handling can affect the correct execution of feeding events, time-sensitive sensing, and actuator control. As a result, one of the main engineering challenges in PLF is to design digital infrastructures that preserve both operational predictability and interoperability across hardware and software layers [12].

Recent research has shown the value of IoT, edge computing, and artificial intelligence for agricultural automation [15–19], but it has also highlighted persistent difficulties in data integration and distributed control [1,15–17,20]. Multi-Agent Systems (MAS) are a useful approach for coordinating distributed components because they support modularity, autonomy, and structured interaction among software entities [21,22]. However, standard agent communication approaches such as Agent Communication Language (ACL) as defined by the Foundation for Intelligent Physical Agents (FIPA) [23] can introduce processing overhead that is problematic for resource-constrained edge devices. In practice, this creates a trade-off: semantically rich communication improves interoperability, but lightweight communication is often required to maintain deterministic and efficient operation in agricultural control systems. This trade-off is particularly relevant in precision feeding, where a functional system must coordinate animal identification, weight acquisition, feeding requests, actuator commands, and data exchange with higher-level decision modules, often under continuous operation. Although several studies have addressed digitalization in livestock systems [23–25], fewer have focused on how to connect semantic agent-based communication with low-overhead control at the device level in a way that is both interoperable and practical for embedded agricultural environments [26].

To address this gap, this paper presents a hybrid multi-agent control architecture for IoT-based precision feeding in pig production. The proposed Controlling Module (CM) combines two complementary communication layers. At the device interface, it uses a lightweight character-delimited TCP/IP protocol to support deterministic exchange with constrained controllers. At the internal software level, it uses an XML-based message structure aligned with FIPA communication primitives [23] to preserve semantic organization for agent interaction. This design seeks to reconcile the competing requirements of real-time operational efficiency and semantic interoperability.

The main contributions of this work are as follows:

- The design of a hybrid Multi-Agent Controlling Module for precision feeding systems that links DSS functions with edge-level farm devices.
- The implementation of a dual-layer communication strategy that separates lightweight device communication from semantically structured internal agent communication.
- The development of a custom serialization/deserialization routine in LabVIEW to process XML-structured agent messages with reduced overhead relative to generic parsing approaches.
- The validation of the proposed architecture through a 120 h laboratory study combining a Digital Twin (DT) simulation of 50 virtual feeders with Hardware-in-the-Loop (HIL) testing of key physical sensing components.

The remainder of the paper is organized as follows. Section 2 describes the architecture of the Controlling Module, the communication design, and the validation procedure. Section 3 presents the validation results. Section 4 discusses the implications, limitations, and relevance of the proposed approach for PLF applications.

2. Related Work

Precision feeding has become a central research line in pig production because it seeks to match nutrient supply to the requirements of each animal in real time, thereby improving nutrient efficiency and reducing the environmental burden of feeding [5–7,9,27]. Pomar and Remus [28] described precision pig feeding as a major step toward sustainability, highlighting its potential to lower nutrient excretion and improve the efficiency of pig production systems. Experimental evidence also shows that feeding growing-finishing pigs with daily tailored diets can reduce standardized ileal digestible lysine intake as well as estimated nitrogen and phosphorus excretion without compromising animal performance or carcass composition. More recent environmental analyses have reinforced this view, showing that precision feeding can reduce the environmental footprint of pig production systems through more efficient use of nutrients [6,8,27].

Alongside nutritional research, automatic feeding technologies have received growing attention as enabling infrastructure for precision livestock systems. AFS for pigs can reduce labor requirements, support remote supervision, and improve the regularity of feed delivery while generating digital records for management and control (Table 1). However, much of this literature focuses on economic feasibility, device operation, or biological outcomes rather than on the software architecture required to connect feeding devices, sensing elements, and higher-level decision modules in a reliable and interoperable way [28–30].

A second relevant body of literature concerns the application of MAS in agriculture and livestock production [22]. Agent-based approaches are well suited to distributed and heterogeneous environments because they enable modularity, autonomy, and coordinated decision-making among multiple software entities. In livestock applications, MAS have been combined with wireless sensor networks to support remote monitoring and embedded intelligence under computationally limited conditions, demonstrating their value for farm environments where devices differ in capabilities and operate under resource constraints (Table 1). Broader agricultural IoT initiatives have also emphasized interoperability and real-time integration of cyber-physical systems, as illustrated by projects such as AFarCloud, which aim to connect agricultural devices, data sources, and decision-support services within distributed digital platforms [31].

At the semantic level, interoperability initiatives such as SAREF4AGRI further show the importance of standardized information models for agricultural IoT (Table 1). SAREF4AGRI was developed to support the agriculture and food-chain domain and includes livestock-oriented use cases and descriptions for sensors, measurements, and decision-support integration [32,33]. These efforts are highly relevant because they improve the interpretability and exchange of information across heterogeneous systems. Nevertheless, semantically rich communication models do not by themselves resolve the timing and processing constraints that arise when low-resource edge controllers must participate in real-time control loops.

This is the gap addressed by the present study. Existing literature supports the value of precision feeding, the usefulness of automatic feeding technologies, and the applicability of MAS and interoperability frameworks in agricultural contexts as summarized on Table 1. Yet fewer studies focus on how to combine lightweight device-level communication with semantically structured internal coordination in a single control architecture for precision feeding applications. The contribution of this work is therefore not another demonstration of the biological value of precision feeding, but the design and validation of a hybrid multi-agent control architecture intended to reconcile semantic interoperability with deterministic operation in a resource-constrained agricultural IoT setting.

Table 1. Summary of works related to this paper.

Study Line	Main Focus	Methods/Architecture	Main Strength	Main Limitation	Position Relative to This Work
Precision pig feeding studies [28–30]	Nutritional precision feeding and sustainability	Nutritional modelling, individualized feeding strategies, biological evaluation	Strong evidence for economic and environmental relevance of precision feeding	Limited detail on interoperable control architecture at device level	Provides the production rationale for the proposed control architecture
MAS for livestock monitoring [22]	Distributed monitoring in livestock systems	MAS with wireless sensors and embedded agents	Shows MAS suitability for heterogeneous and constrained livestock environments	Focused on monitoring rather than deterministic feeding control	Supports the agent-based design rationale of the present study
MAS/IoT agriculture approaches [31]	Intelligent multi-agent agricultural control	IoT plus multi-agent coordination	Demonstrate the value of distributed intelligence in agriculture	Not specific to precision feeding or FIPA-aligned internal communication	Conceptually related, but different application and control requirements
Agricultural intelligent agent [34]	Methodological frameworks for agricultural AI	Three-dimensional integrated agent framework	Up-to-date, holistic taxonomy of agent technologies across full agricultural value-chain scenarios	Conceptual focus; lacks low-level edge-device control or real-time determinism	Conceptual background on intelligent agents, but not a concrete architecture for precision livestock feeding
Agricultural interoperability frameworks [32]	Semantic interoperability across agri-food systems	Ontologies, reference models, data exchange frameworks	Strong interoperability foundation across heterogeneous platforms	Do not by themselves solve real-time constraints in low-resource controllers	Motivates the semantic layer of the proposed architecture

3. Materials and Methods

3.1. System Overview and Role of the Controlling Module

The CM was designed as the software component responsible for supervising and coordinating the operation of the AFS devices used in the precision feeding platform. Its main role was to connect the DSS, which calculated animal feeding requirements, with the physical layer composed of feeding units, weighing devices, and associated sensors.

Within this architecture, the CM managed device communication, supervision tasks, and real-time data handling. It therefore acted as an intermediate control layer between high-level nutritional decision-making and low-level device execution. This role was especially relevant because the feeding platform combined heterogeneous hardware components that had to exchange information reliably and within operational time constraints.

3.2. Multi-Agent Design and Internal Coordination

The CM was implemented as a MAS in which control tasks were distributed across specialized software agents. This design was adopted to separate communication, supervision, data handling, user interaction, and diet execution into modular components with clearly defined responsibilities [21]. Compared with a monolithic control process, this structure simplified maintenance and allowed the system to manage multiple AFS devices concurrently.

The CM included nine specialized agents, classified as either Simple Reflex or Model-Based Reflex. The distinction was based on the information required to perform each task. Simple Reflex agents were assigned to tasks that depended mainly on the current input and required rapid, deterministic responses with minimal internal state. Model-Based Reflex agents were assigned to tasks that required state retention, context awareness, or supervision over time, such as connection management, anomaly tracking, and data persistence. Table 2 summarizes the agent set, its classification and the main function assigned to each agent. In practical terms, the Distribution, Diet Strategy, UI Management, RT Monitor, and API Connector agents were implemented as Simple Reflex agents because they mainly responded to incoming messages or user actions according to predefined rules. Their operation followed a direct input-response pattern, which reduced processing overhead and supported timely execution in the control loop. By contrast, the Communication, Supervision, Data Management, and DSS Activation agents were implemented as Model-Based Reflex agents because they had to preserve operational context across events, including device status, stored measurements, system incidents, and decision triggers.

Table 2. Software agents of the Controlling Module, their architecture type, and their main operational function within the precision feeding system.

Agent Name	Agent Type	Functionalities
Communication	Model-Based Reflex	Manages device connections and encodes/decodes external messages
Distribution	Simple Reflex	Validates and routes incoming and outgoing messages
Supervision	Model-Based Reflex	Monitors system status, watching events and abnormal conditions
Diet strategy	Simple Reflex	Converts DSS outputs into operational feed parameters
Data management	Model-Based Reflex	Stores raw data and metadata and provides internal data support
UI management	Simple Reflex	Displays diet configuration and system status to the user
RT monitor	Simple Reflex	Collects real-time device information for monitoring
DSS activation	Model-Based Reflex	Determines when DSS execution should be triggered
API connector	Simple Reflex	Exchanges system data configuration with the external REST API.

Abbreviations: UI: user interface; RT: real time; DSS: decision support system; API: application programming interface.

Coordination among agents was achieved through structured message passing and controlled access to shared internal data. Incoming messages from devices were first received by the Communication agent, decoded, and converted into an internal structured representation. The Distribution agent then routed each validated message to the corresponding target agent or agents according to message type and destination. This workflow reduced ambiguity in message handling and ensured that each functional task was processed by the appropriate component.

Synchronization inside the CM relied on the execution model of LabVIEW and the use of Functional Global Variables (FGVs) for controlled data exchange between agents. FGVs were used as protected shared memory elements with predefined operations such as initialization, read, write, and verification. This approach limited uncontrolled concurrent access to shared data and reduced the risk of race conditions during agent interaction [35,36]. Conflict resolution was therefore handled through message validation, sequential routing logic, and controlled access to shared internal variables rather than through negotiation-based arbitration.

The core control loop was organized around four main agents: Communication, Distribution, Supervision, and Diet Strategy. The Communication agent managed connection establishment and message encoding/decoding between the CM, the DSS, and field devices. The Distribution agent checked message integrity and routed messages to other internal agents. The Supervision agent monitored device status and generated alerts when abnor-

mal conditions, disconnections, or faults were detected. The Diet Strategy agent translated nutritional decisions received from the DSS into operational parameters, including feed dose and latency time, for the AFS units.

Additional agents supported these core functions. The Data Management agent stored raw measurements and metadata and provided configuration data when required by other agents. The RT Monitor agent collected real-time device status information for monitoring, while the UI Management agent presented this information to the operator. The DSS Activation agent determined when the external DSS should be triggered, and the API Connector managed data exchange with an external REST API. Together, these agents extended the CM beyond device control to include supervision, traceability, and integration with external software services. The overall organization of these agents and their interactions with peripheral subsystems is summarized in Figure 1.

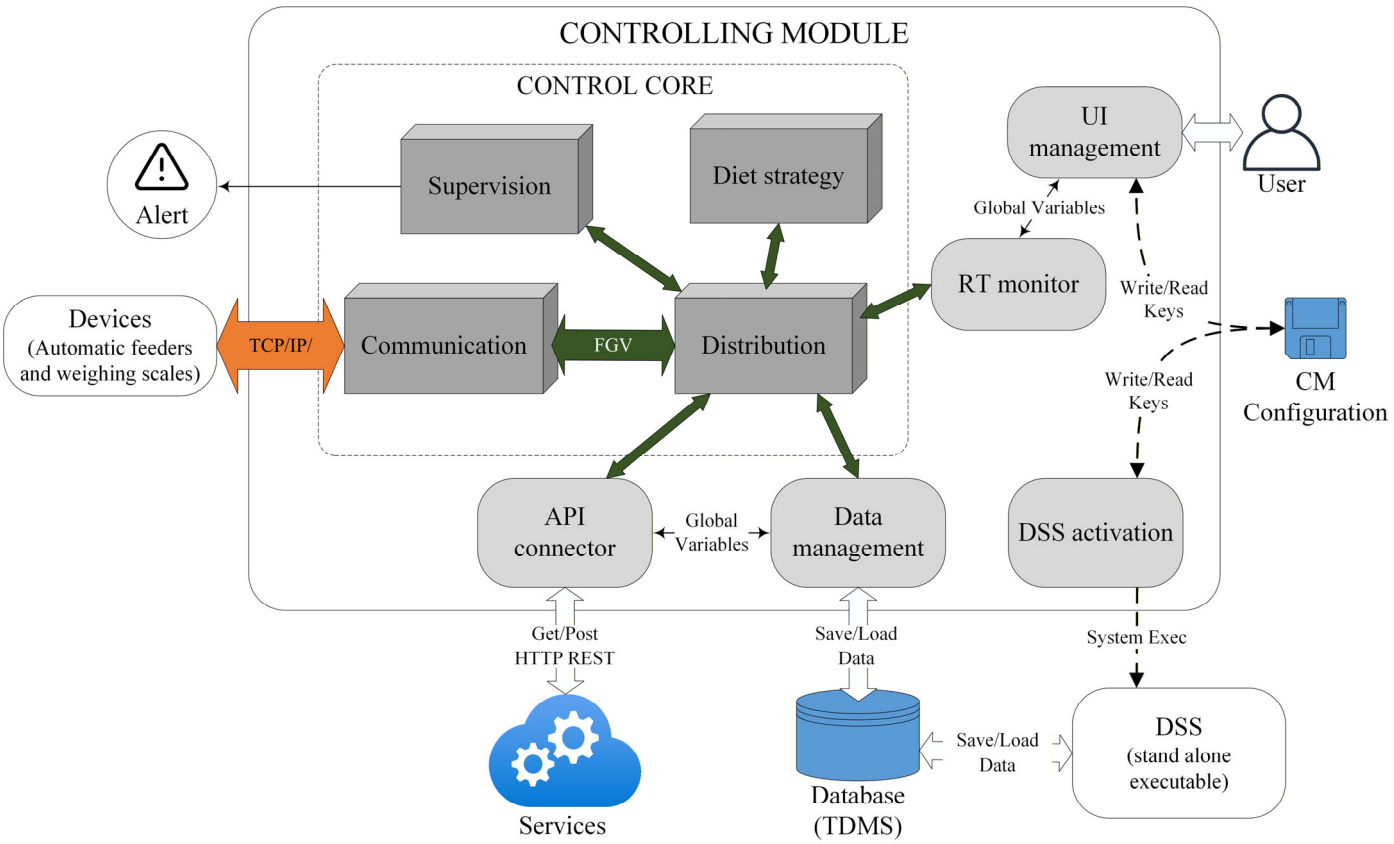


Figure 1. Internal structure of the Controlling Module showing the core Control agents (Communication, Distribution, Supervision and the Diet Strategy) and their connections with auxiliary agents for user interaction, real-time monitoring, data management, API exchange and DSS activation in the precision feeding system.

Fault tolerance was addressed at a practical architectural level rather than through formal verification. First, the modular design reduced failure propagation by isolating functions across specialized agents. Second, the Supervision agent continuously monitored connection status and abnormal operating conditions through a watchdog mechanism. Third, communication recovery procedures were implemented to restore operation after network interruption events. Although the current work did not include a formal deadlock analysis or theorem-based verification of agent interactions, the use of controlled message routing, non-reentrant shared access patterns, and supervisory monitoring reduced the likelihood of uncontrolled blocking conditions during operation.

The scalability of the CM depends mainly on the number of connected devices, message frequency, message size, and the computational resources available to execute concurrent loops. In the present study, the architecture was empirically validated with 50 virtual devices because this represented the target laboratory deployment scenario and provided a demanding communication load for the available infrastructure. Beyond this scale, performance is expected to depend on the cumulative scheduling burden of concurrent connections and the rate at which messages must be parsed, routed, stored, and displayed. For this reason, the current manuscript presents scalability up to the validated range and discusses larger-scale deployment as future work rather than as a demonstrated property. To make the interaction between agents more transparent, the main steps of the internal message-processing loop are listed in Algorithm 1.

Algorithm 1. Simplified message-processing sequence in the CM.

1. Receive message from device or DSS.
2. Communication agent validates connection and decodes the incoming string.
3. Convert message to internal structured format.
4. Distribution agent checks syntax and message class.
5. Route message to one or more target agents.
6. Target agent executes task, for example supervision, diet update, storage, visualization, or API transfer.
7. If a fault or abnormal condition is detected, the Supervision agent generates an event or alert.
8. If a device response is required, the Communication agent encodes and sends the output message.

3.3. Communication Architecture

Communication in the proposed architecture was organized in two complementary layers: an external device-to-CM layer and an internal agent-to-agent layer. This separation was introduced because the system had to satisfy two different requirements at the same time: efficient communication with resource-constrained AFS devices [35], and structured information exchange among software agents inside the CM. Table 3 summarizes these two communication layers and their respective implementation choices.

Table 3. Communication layers implemented in the Controlling Module, distinguishing the external interface with Automatic Feeding System devices from the internal interface used for communication among software agents.

Communication	Physical Transport Layer	Software Agency Layer	Main Purpose
AFS device ⇔ CM	TCP/IP	Character-delimited ASCII stream	Deterministic and lightweight external communication
Agent ⇔ Agent	Functional Global Variables (FGVs) in LabVIEW	XML-based structured internal message	Internal coordination, traceability and semantic organization

Abbreviations: ASCII: American Standard Code for Information Interchange; CM: Controlling Module; FGV: Functional Global Variable; TCP/IP: Transmission Control Protocol/Internet Protocol; XML: extensible markup language.

3.3.1. External Device–CM Communication

At the external layer, communication between field devices and the CM was implemented over TCP/IP through Ethernet or Wi-Fi. The CM acted as the server and established communication with each device individually. At the application level, messages were exchanged through a lightweight character-delimited ASCII protocol designed to reduce parsing complexity in embedded controllers. This design choice prioritized deterministic

handling of frequent control and monitoring messages over full protocol generality. Each external message followed a fixed four-part structure: Start, Identification, Content, and End. The Start field indicated message origin, the Identification field contained routing and traceability information, the Content field carried the operational payload, and the End field marked message completion. This rigid structure simplified validation and reduced ambiguity during message decoding. Table 4 provides an example of a consumption message exchanged between an AFS device and the CM.

Table 4. Example of an external consumption message sent from an AFS device to the Controlling Module (CM) using the character-delimited protocol.

Section	Item	Description
Start	${	Beginning of the message. "$" indicates that the direction of the message goes from the device to CM.
Identification	<18.04>	Receiver (destination address) of the message (18: type of agent; 04: number of agent of that type)
	<FAB004>	Sender of the message (F: Feeder; AB004: feeder identification)
	<00016>	Number of the message
	<2024/02/24>	Date of the message
	<12:50:18>	Local time of the message
	<02>	Class level of the message (02: animal consumption)
	<1>	Level of priority (1: normal)
	<Feeder>	Type of device involved in the conversation with the CM
Content	<00274>	Tracking number of the consumption message
	<2024/02/24>	Date of the consumption
	<12:49:00>	Local time IN of the message
	<12:50:17>	Local time OUT of the message
	<724000000000025>	Animal identification
	<011,0>	Motor 1 base run time, in seconds
	<003,0>	Motor 2 base run time, in seconds
	<00033,0>	Motor 1 total run time, in seconds
	<00009,0>	Motor 2 total run time, in seconds
	<003>	Services activated for motor 1
	<003>	Services activated for motor 2
End	} \CRLF	End of the message

Abbreviations: CM: Controlling Module; CRLF: Carriage Return and Line Feed.

The use of a lightweight external protocol was motivated by practical implementation constraints. Standard agent-oriented syntaxes can be too verbose for low-resource controllers when message exchange must occur continuously and with predictable timing. For that reason, semantic richness was not enforced directly at the physical device interface; instead, it was introduced at the internal software layer after message reception and validation.

3.3.2. Internal Agent–Agent Communication

Inside the CM, agent-to-agent communication (Table 3) relied on structured data exchange supported by FGVs and a standardized internal message format. FGVs were used to preserve state, control access to shared data, and coordinate interactions among LabVIEW virtual instruments [37,38]. This mechanism supported synchronization by restricting read and write operations to predefined actions such as initialization, read, write, and verification (supplementary Figure S1). In this way, internal communication combined structured routing with controlled shared-state access [38]. After reception at

the physical layer, external messages were converted into typed internal data structures and then represented in an XML-based format aligned with Foundation for Intelligent Physical Agents (FIPA) [23] Agent Communication Language concepts. The purpose of XML in this architecture was not to optimize transmission efficiency, but to provide a clear and extensible internal representation for agent communication, message traceability, and integration with higher-level software logic. This choice improved readability and structure inside the CM while preserving separation from the lightweight device protocol.

A custom serialization and deserialization routine was implemented in LabVIEW to process this internal XML structure. The routine was tailored to the predefined message schema used in the CM rather than designed as a general-purpose XML parser. This reduced processing overhead inside the application and ensured direct conversion into strict LabVIEW clusters. However, the present study does not claim a formal benchmark against JSON, binary serialization, DOM, or SAX parsers; therefore, the XML choice should be interpreted as an implementation decision aligned with the needs of the proposed architecture rather than as a universally superior format.

Figure 2 illustrates the internal processing implemented by the Communication agent. The workflow comprised three stages: acquisition of the raw byte stream, validation and conversion into the internal structured message format, and clustering into typed LabVIEW data objects for subsequent routing and agent processing. Table 5 presents an example of the internal XML representation generated from a device message.

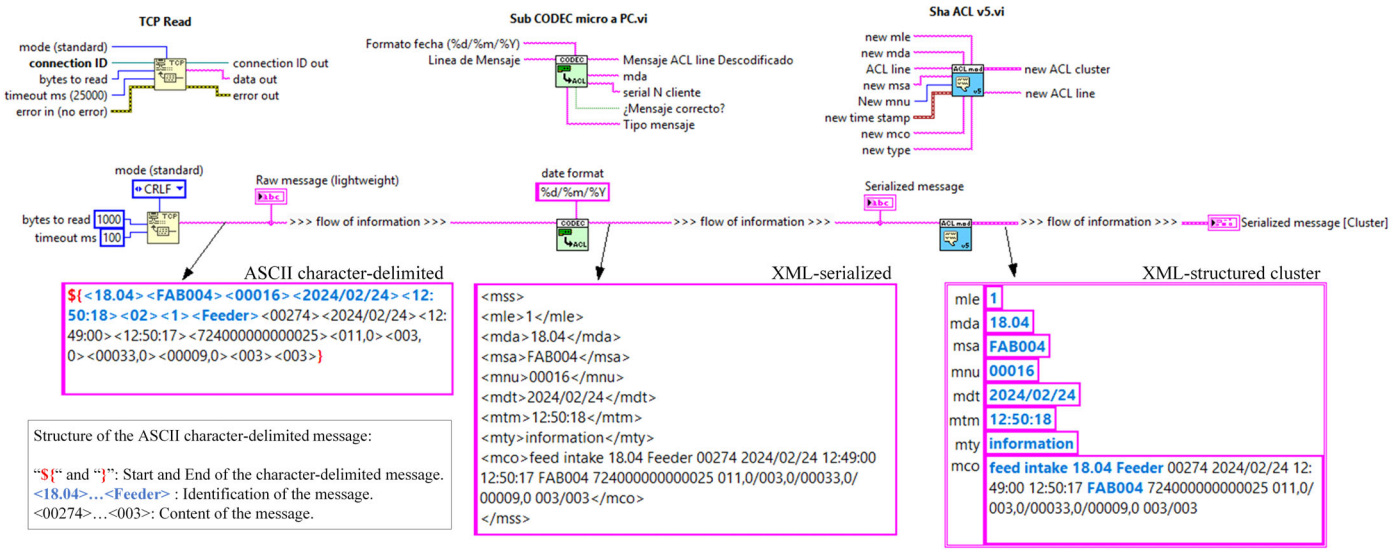


Figure 2. Internal processing pipeline of the Communication agent. The diagram shows three main stages: acquisition of the TCP/IP byte stream from the device, transformation into a validated XML-based internal message, and deserialization into strictly typed LabVIEW data structures before routing to the decision-making agents.

Security was addressed at a basic architectural level. Message exchange relied on controlled device-to-server connections, internal message validation, and supervision of abnormal connection events. These mechanisms contributed to operational safety by reducing malformed message propagation and by supporting fault detection during communication loss or device disconnection. Nevertheless, the current prototype did not implement end-to-end encryption, formal authentication, or attack-resilience testing, and these elements should therefore be considered part of future development rather than demonstrated features of the present system.

Table 5. Example of the internal XML-based representation of a consumption message after conversion by the Communication agent.

Item	Description
<mss>	Start message markup
<mle>1</mle>	Message level (1: low priority)
<mda>18.04</mda>	Message destination agent (receiver) address (18: Communication agent; 04: # device connection)
<msa>FAB004</msa>	Message sender agent address (FAB004: Serial number of the device)
<mnu>00274</mnu>	Number of the message
<mdt>2024/02/24</mdt>	Local date of the message
<mtm>12:50:18</mtm>	Local time of the message
<mty>information</mty>	Message type
<mco> feed intake 18.04 Feeder 00274 2024/02/24 12:49:00 12:50:17 FAB004 724000000000025 011,0/003,0/00033,0/00009,0 003/003 </mco>	Content of the message
</mss>	End message markup

3.4. Development Environment and Implementation

The CM was implemented in National Instruments LabVIEW Professional software [39]. This platform was selected because the study required tight integration between software control logic, communication routines, real-time monitoring, and hardware-oriented testing elements. In addition, the dataflow execution model of LabVIEW facilitated the organization of parallel tasks such as message acquisition, routing, supervision, logging, and user-interface updates within the same application [35,39].

The choice of LabVIEW should be interpreted as an implementation decision associated with the prototype requirements rather than as a general claim of superiority over other environments such as Python, C++, or ROS [36,39]. Other platforms could also support the development of distributed control software, but LabVIEW provided a practical advantage in this study because it allowed direct integration of communication, instrumentation, and visualization components in a single development environment [36]. This was particularly useful for rapid prototyping and for HIL validation, where close interaction with acquisition devices and control-oriented routines was required.

The implementation also used LabVIEW Functional Global Variables [37] and virtual instruments (VIs) to organize state management, inter-agent coordination, and modular execution. These elements supported the modular structure of the CM and simplified the separation between communication tasks, supervisory functions, storage routines, and operator-level visualization. In this sense, the platform was consistent with the architectural objective of distributing the control logic into specialized software components. To complement the long-duration validation with a controlled benchmark of the internal communication mechanism, four stress tests were conducted on the FGV-based message distribution sub-routine. A dedicated VI generated synthetic messages by randomly selecting among the 12 message types expected during normal operation (mainly watchdog, consumption, weight, and status messages). The message generation rate was progres-

sively increased from 100 to 1000 messages per second by shortening the loop period of the generator routine, and each rate was maintained for 5 min to ensure homogeneous exposure and direct comparability across tests. Two receiving-agent configurations were evaluated: a "fast" case with an internal loop period of 1 ms, representing agents with minimal additional processing, and a "slow" case with a 10 ms loop, representing agents temporarily occupied with more demanding internal tasks. For each loop configuration, tests were performed with either a single receiving agent or six receiving agents sharing the same total traffic load. In all cases, the average number of messages stored in the internal communication buffer was recorded at each rate to identify the onset of queue accumulation.

The experimental validation was intentionally performed on a low-specification computer running Windows XP with 2 GB of RAM. This hardware environment was not selected to represent a modern deployment standard, but to test whether the CM could maintain stable operation under constrained computational resources. Using an outdated platform therefore provided a conservative validation scenario for CPU load, memory consumption, and long-duration execution stability. The goal was to show that the proposed architecture did not depend on high-end hardware to sustain the laboratory-scale workload considered in this study. This choice also imposes an important limitation. The Windows XP environment does not reflect current cybersecurity, maintainability, or deployment standards for production agricultural systems. For that reason, the results obtained in this hardware setting should be interpreted as evidence of computational feasibility under constrained resources, not as a recommendation for operational deployment on obsolete operating systems. Future implementations should therefore migrate the architecture to supported operating systems and contemporary hardware platforms before field-scale adoption.

No formal benchmark against alternative software frameworks was conducted in the present study. Consequently, claims regarding reduced development complexity, faster implementation, or superior runtime performance relative to Python, C++, or ROS cannot be made from the available evidence. The contribution of this section is therefore limited to documenting the implementation environment actually used and explaining why it was suitable for the prototype and validation tasks reported here.

3.5. Validation Procedure

The validation of the CM was designed as a laboratory-based assessment of communication reliability, control logic, and computational performance under sustained operation. The procedure combined two complementary environments: (i) a DT environment for large-scale communication and concurrency stress testing, and (ii) a HIL environment for testing selected physical interactions with real devices and sensors. This combined design was intended to evaluate both software-level scalability and the correct handling of critical field signals before on-farm deployment.

The DT environment was implemented as a dedicated LabVIEW application running on a computer independent from the CM. It simulated the concurrent operation of 50 virtual AFS units, each reproducing the communication behavior, firmware logic, and state transitions required for interaction with the CM. The purpose of this environment was to generate sustained message traffic and repeated control events over a continuous five-day test period (120 h). In this way, the DT was used as a communication and control emulator rather than as a full biological or farm-environment replica. Accordingly, the DT validation should be interpreted within a bounded scope. It assessed the ability of the CM to manage simultaneous virtual devices, message routing, recovery from communication interruptions, and long-duration execution under load. However, it did not reproduce all sources of variability present in commercial farms, such as animal behavioral variability,

changing environmental conditions, heterogeneous network infrastructures, or management differences across farms. For this reason, the DT results demonstrate laboratory-scale technical feasibility, but not full field equivalence.

In parallel, a HIL configuration was constructed to test selected physical interfaces that were critical for system operation. This setup used four prototype AFS controller boards equipped with the complete firmware and communication functions, connected to four RFID antennas and four PNP inductive sensors. The HIL setup was specifically intended to validate animal identification events, feed-demand sensing, and the corresponding event generation and communication logic. Unlike the DT environment, which focused on scale and concurrency, the HIL configuration focused on signal-level realism for key sensing functions.

To emulate repeated identification and demand events in a controlled and reproducible way, a laboratory mechanical arrangement was used. RFID tags and metallic targets were mounted on a rotating wheel so that stopping positions generated synchronized RFID detection and inductive sensor activation at predefined points. This configuration allowed repeated testing of identification changes, sensor triggering, and the associated control responses without requiring live animals. The HIL system and the DT environment operated concurrently during the 120 h validation period.

During the validation period, computational performance of the CM was monitored continuously using the NI LabVIEW Profile Performance and Memory tool. Execution time and memory usage of the main virtual instruments were recorded to identify bottlenecks and to detect abnormal resource growth during prolonged operation. CPU usage, memory consumption, and internal communication-buffer behavior were also observed as part of the stress test. These measurements were intended to verify computational feasibility under the tested workload rather than to provide a benchmark against alternative software platforms. The validation design also has clear limitations. First, the study was conducted in a laboratory setting and did not include testing across multiple farms or production datasets. Second, scalability was empirically assessed up to 50 virtual devices, which represented the target validation workload of the prototype, but larger deployments were not experimentally verified. Third, the HIL setup covered selected physical events only and did not reproduce the full mechanical and environmental complexity of commercial AFS installations. Therefore, the present validation supports the technical soundness of the proposed architecture at laboratory scale, while broader external validation remains future work.

4. Results

The validation campaign was conducted over a continuous 120 h laboratory test period using the DT and HIL configurations described in Section 3.5. The results are summarized in Table 6 and are presented here according to the three evaluated dimensions: communication and control reliability, physical sensing validation, and computational performance.

Table 6. Summary of the outcomes obtained during the 120 h validation campaign. The table reports the observed behavior of the Controlling Module under Digital Twin and Hardware-in-the-Loop (HIL) tests, together with computational performance indicators measured on the implementation hardware.

Test	Condition	Outcome	Repetition (Success/Failure)
Communication Reliability (50 virtual feeders)	CM connected with feeder via Wi-Fi.	Yes	1200/0
	Automatic reconnection when Access Point is switched off and back on.	Yes	1200/0
	Automatic reconnection when Feeder is switched off and back on.	Yes	1200/0

Table 6. *Cont.*

Test	Condition	Outcome	Repetition (Success/Failure)
Communication Reliability (50 virtual feeders)	Automatic reconnection when Ethernet cable is disconnected and reconnected.	Yes	1200/0
	Watchdog messages successfully sent.	Yes	120 messages in 1 h (0 lost)
	Feeder's system clock synchronized with CM's clock.	Yes	2000
Feed Formulation Management (50 virtual feeders)	Formula request message sent by Feeder upon animal detection.	Yes	560/0
	CM sends the animal's formula (composition).	Yes	560/0
	All formulas are deleted when the "ghost animal" (NNN) formula is sent to the feeder.	Yes	N/A
	Feeder reconnection with CM triggers an NNN formula request.	Yes	N/A
	Feeder retains formulas in memory after a restart.	Yes	N/A
Consumption Data Handling (50 virtual feeders)	Consumption data for an identified animal transmitted when tag is no longer detected.	Yes	2590/0
	Consumption data lines are stored in feeder memory when communication is lost.	Yes	N/A
	Restricted formula erased upon successful transmission of consumption data.	Yes	N/A
	Consumption data is sent when the animal's identification changes.	Yes	N/A
Fault & Disconnection Events (50 virtual feeders	Event message sent upon detection of an electric overload fault.	Yes	N/A
	Event message sent upon detection of a short circuit fault.	Yes	N/A
	Event message generated by disconnection of feeders or devices.	Yes	N/A
Computer performance (50 virtual feeders)	Average CPU usage	15% (max: 17%; min: 14%)	N/A
	Average Memory usage	214 MB (max: 350 MB; min: 166 MB)	N/A
	Internal buffer	0% (max: 4%)	N/A
Identification & Sensing (4 RFID antennas + 4 inductive sensors)	Identification message sent by Feeder.	Yes	N/A
	Latency timer activated upon animal detection.	Yes	N/A
	Demand sensor disabled while latency time is active.	Yes	N/A
	Event messages are sent when demand sensor is triggered.	Yes	N/A
	Event message sent when motors are running.	Yes	N/A
	New latency time activated when motors stop.	Yes	N/A
	Event message sent when animal identification changes.	Yes	N/A

4.1. Communication and Operational Tests

Under the DT configuration, the CM maintained stable communication with 50 virtual AFS units throughout the 120 h test period. Reconnection tests associated with temporary communication interruptions, including access point shutdown, device restart, and Ethernet disconnection, were completed in all tested repetitions reported in Table 6. Likewise, watchdog transmission, time synchronization routines, formula request handling, and consumption-data transfer events were completed without recorded loss in the monitored test runs. These observations indicate that the proposed architecture maintained consistent communication behavior under the tested laboratory workload. However, the absence of observed failures in this experiment should be interpreted as a result limited to the tested conditions and duration, not as proof that failures are impossible in broader deployment scenarios. In the same way, the reconnection results support the presence of effective recovery behavior under simulated interruptions, but they should not be generalized beyond the tested communication conditions.

The data-handling routines also remained stable during stress testing. Consumption records were retained during temporary communication loss and subsequently transmitted when connectivity was restored, while formula management events, including formula deletion and reconnection-triggered requests, followed the expected control logic defined for the CM. These results support the internal consistency of the message-handling and state-management routines under sustained load.

The stress tests (see Figure 3) showed that the FGV-based internal communication mechanism maintained low and bounded buffer occupancy across the full range of tested message rates when the receiving agents operated in the fast configuration with a 1 ms loop period, both for one and for six agents. Under these conditions, the average number of messages stored in the buffer remained close to one, even when the generation rate reached 1000 messages per second. When the receiving agents were configured with the slower 10 ms loop, intended to emulate additional internal processing tasks, queue growth appeared at lower transfer rates, and unbounded accumulation was observed only at the highest tested rates, particularly in the single-agent case. Because each message rate was applied for 5 min in all configurations, these observations can be directly compared across tests despite the increasing number of messages generated at higher rates. In practical terms, the saturation points remained far above the communication demand observed in a preliminary field deployment, where the average daily traffic per feeder was approximately 23,000 messages, indicating that the implemented internal communication mechanism provides a substantial operational margin with respect to the traffic levels expected in commercial pig fattening farms.

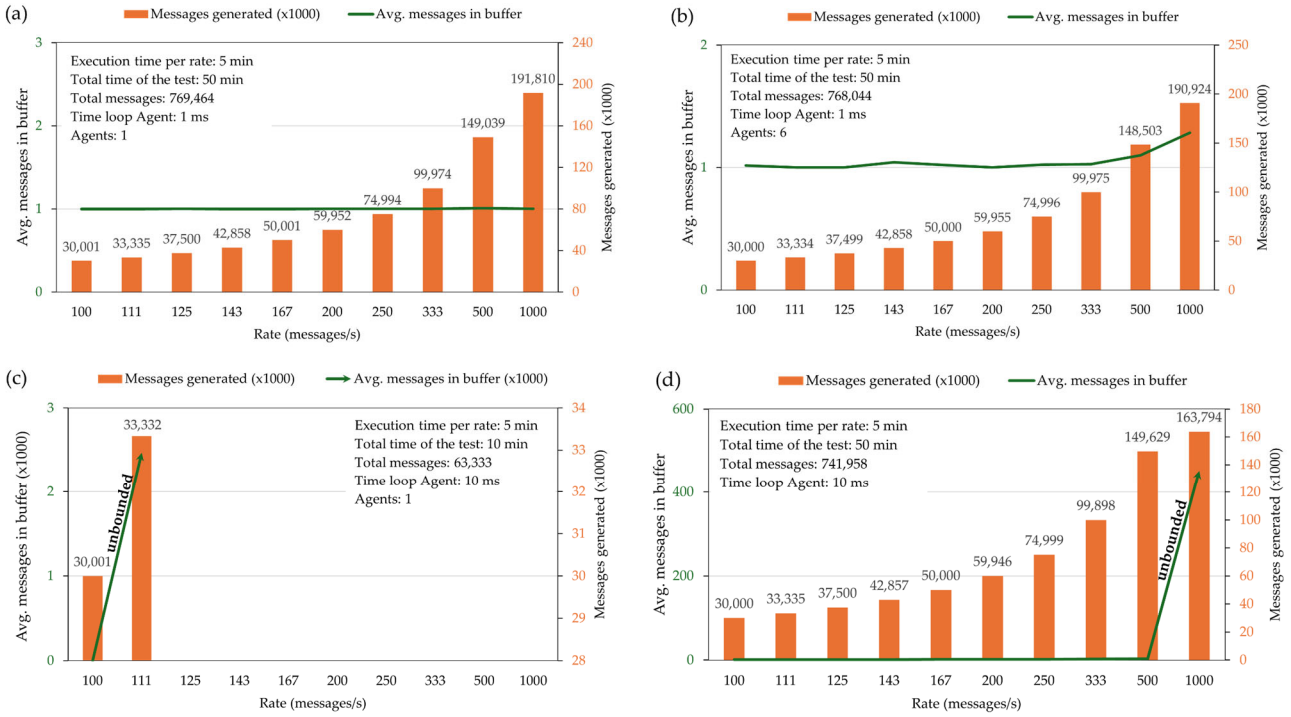


Figure 3. Stress tests of the FGV-based internal communication mechanism under increasing synthetic message rates. (**a**) Single-agent configuration with a 1 ms agent loop: total number of generated messages (bars, ×1000) and average number of messages stored in the internal buffer (line, ×1000) for each rate, with each rate applied for 5 min. (**b**) Six-agent configuration with a 1 ms agent loop under the same sequence of rates, showing that buffer occupancy remains low and bounded across the tested range. (**c**) Single-agent configuration with a 10 ms agent loop, where unbounded buffer accumulation occurs at the highest tested rates, illustrating the most demanding case for the internal communication mechanism. (**d**) Six-agent configuration with a 10 ms agent loop, representing agents with additional internal processing; buffer growth appears only at the highest tested rates, where the average number of queued messages increases markedly.

4.2. HIL Sensing Tests

The HIL configuration confirmed the correct operation of the sensing and event-generation logic for the physical interfaces included in the laboratory setup. Across the test period, the prototype boards correctly processed RFID-based identification events,

inductive demand-sensor activation, latency-timer triggering, and motor-related event messages as specified in Table 6. Event generation associated with changes in animal identification and demand-sensor status was also observed as expected.

These results indicate that the CM correctly handled the critical sensing signals reproduced in the HIL configuration. At the same time, the interpretation of these results should remain bounded to the tested setup, which reproduced selected physical events using laboratory hardware rather than the full variability of commercial farm operation.

4.3. Computational Performance

Performance profiling of the CM was conducted during the same 120 h validation period using the NI LabVIEW Profile Performance and Memory tool. Under the workload generated by 50 virtual AFS units, average CPU usage was 15%, with observed values ranging from 14% to 17%, while average memory use was 214 MB, with a maximum of 350 MB and a minimum of 166 MB. Internal communication-buffer occupancy remained at 0% on average, with a maximum observed value of 4%.

At the subroutine level, the profiling analysis identified Sha Convert Date-Time strings to Seconds.vi as the virtual instrument with the highest cumulative execution time, with 2656 ms accumulated over 114,142 calls. The highest observed memory allocation at the VI level was associated with FGV agents.vi, with a peak of 5.69 kB. These measurements indicate that the prototype remained computationally stable during prolonged operation on the tested hardware platform. However, because no benchmark against alternative implementations or hardware configurations was performed, these values should be interpreted as descriptive profiling results for the present prototype rather than comparative performance evidence.

5. Discussion

The results support the technical feasibility of the proposed CM as a laboratory-scale control architecture for interoperable precision feeding. Across the 120 h validation campaign, the system maintained stable communication with 50 virtual AFS units, correctly handled key HIL events, and operated with limited computational resource use on constrained hardware. These findings indicate that the architecture is sound under the tested conditions, while broader claims regarding full field deployment should remain outside the scope of the present study.

5.1. Architectural Soundness Through Multi-Agent System Design

The MAS design contributed to the modular organization of communication, supervision, diet management, storage, and interface functions within the CM. Distributing these functions across specialized agents reduced dependence on a single monolithic control routine and supported clearer separation of responsibilities inside the software architecture [40]. In this sense, the MAS approach was useful not only as a conceptual framework but also as a practical way to manage heterogeneity and concurrency in the precision feeding environment.

The validation results are consistent with that architectural choice. Stable message handling, recovery after tested communication interruptions, and low buffer occupancy suggest that the internal coordination logic was adequate for the workload examined in this study. However, these observations should be interpreted as evidence of architectural soundness under the tested laboratory conditions, not as proof of general fault tolerance for all deployment scenarios or scales.

5.2. Interoperability and Constrained Devices

One of the central design challenges addressed in this study was the tension between semantic interoperability and the operational constraints of low-resource field devices [41–43]. Standard agent communication approaches are useful for structured interaction, but their direct use at the device interface may introduce unnecessary overhead in real-time agricultural control systems with constrained embedded hardware. The proposed dual-layer strategy was intended to address this issue by separating lightweight external communication from richer internal software representation.

From this perspective, the contribution of the CM is not the replacement of existing standards nor a formally verified coordination model, but their selective adaptation to the requirements of AFS-oriented control. The external character-delimited protocol reduced communication complexity at the device interface, whereas the internal XML-based structure preserved traceability and structured agent communication inside the CM. Each agent is associated with a unique destination address, which the FGV-based communication sub-routine uses to determine the target of each message and to organize routing in a deterministic way. Although the present work does not include formal deadlock analysis or model checking, the stress tests (Figure 3) show that the internal communication mechanism can operate with low and bounded buffer occupancy under message rates far above those expected in the target application, particularly when the receiving agents are not delayed by additional internal processing. Even in the slower 10 ms configuration, unbounded queue growth was observed only under synthetic message rates that remain well above the communication demand observed in a preliminary farm deployment, where each feeder is expected to generate on average approximately 23,000 messages per day. These results support the practical adequacy of the implemented coordination approach and suggest that the risk of uncontrolled blocking is low under the intended operating conditions, while more formal verification of the multi-agent interactions remains an important line of future work. This arrangement appears suitable for environments in which communication efficiency and software organization must coexist, although the present study did not include a formal benchmark against JSON, binary schemes, or alternative agent platforms [44–46].

Beyond the comparison with FIPA ACL [23], it is relevant to position the proposed architecture against foundational agent frameworks like KQML (Knowledge Query and Manipulation Language) and its associated content language, KIF (Knowledge Interchange Format). Historically, KQML/KIF was designed to enable knowledge sharing through First-Order Logic sentences [47], allowing agents to perform complex logical inferences.

However, in the context of Precision Feeding Systems, the requirement is not logical deduction but deterministic control. Transmitting operational parameters (e.g., feed rations, weight matrices) using the declarative logic syntax of KIF would introduce unnecessary abstraction and parsing overhead. The XML-based implementation represents a shift from the 'Symbolic AI' paradigm of KQML, where agents exchange logical theorems, to a 'Data-Centric' paradigm suitable for Industry 4.0, where agents exchange strictly typed, actionable data structures. This ensures that the Control Module maintains the semantic intent of the communicative act (the essence of KQML) while utilizing a syntax optimized for the rigorous data integrity standards of modern automation. Table 7 compares the standard KQML/KIF and FIPA ACL specifications with the adapted implementation used in the CM.

Table 7. Conceptual comparison between classical agent communication frameworks (KQML/KIF, FIPA ACL) and the Controlling Module (CM) implementation. The table highlights differences in syntax, message performatives, and primary design goals with respect to the real-time requirements of the present system.

Feature	KQML/KIF (Foundational Academic)	FIPA ACL (Interoperability Standard)	CM Implementation (Industrial Optimization)
Syntax/Encoding	LISP-like S-expressions. Nested parentheses difficult to parse in real-time.	LISP-like S-expressions. Text-based strings requiring significant parsing overhead.	XML-Serialized. Managed by a custom serialization algorithm to minimize parsing overhead while ensuring strict typing.
Physical Transport	KIF (Knowledge Interchange Format). Based on First-Order Logic (symbolic reasoning).	Flexible (e.g., FIPA-SL). Allows various formats but often defaults to semantic languages.	Structured Data Types. Operational values (Integers, Floats) optimized for control loops.
Message Performative	Open Set. Extensible but often ambiguous (e.g., advertise, broker).	Standardized Set. Fixed set of communicative acts (e.g., INFORM).	Mapped Attribute Codes. Internal codes mapped to FIPA acts for efficiency (e.g., <02> → INFORM).
Primary goal	Knowledge Sharing. Enabling agents to exchange complex logical theorems.	Open Interoperability. Allowing heterogeneous software systems to interact.	Real-Time Determinism. Ensuring type safety and low latency for hardware control.

Abbreviations: ACL: Agent Communication Language; ASCII: American Standard Code for Information Interchange; CM: Controlling Module; FIPA: Foundation for Intelligent Physical Agents; FGV: Functional Global Variable; KIF: Knowledge Interchange Format; KQML: Knowledge Query and Manipulation Language; LISP: List Processing Language; SL: Semantic Language; TCP/IP: Transmission Control Protocol/Internet Protocol; XML: Extensible Markup Language.

The CM uses a purpose-built serialization/deserialization routine tailored to the predefined internal message schema. Unlike generic XML parsers that must account for unknown schemas, our custom implementation was strictly coupled to the agent ontology. This allowed for deterministic deserialization, where the incoming text stream is mapped directly to operational variables (e.g., animal consumption) with minimal CPU cycles. This approach aligns with the recommendations of Leitão [44] regarding the adaptation of IT standards for real-time manufacturing, demonstrating that semantic interoperability can be achieved without sacrificing the performance required by the hardware control layer. Pěchouček and Mařík [48] emphasize that in successful industrial agent deployments, such deviations from strict syntactic standards are often required to ensure operational robustness. Despite these syntactic adaptations, the system preserves semantic interoperability. By explicitly mapping message fields to communicative acts (e.g., Inform, Request) and shared ontologies, the architecture satisfies the fundamental requirement for agent cooperation defined by Genesereth and Ketchpel [49]. This hybrid approach effectively bridges the gap, ensuring that data from constrained agricultural IoT devices can be semantically interpreted and acted upon by advanced DSSs.

5.3. Relevance for Automatic Feeding Systems

The study also contributes to the digital control layer needed for Automatic Feeding Systems in precision livestock farming. In practical terms, individualized feeding depends not only on nutritional models and sensing technologies, but also on a control architecture capable of receiving identification events, requesting formulas, handling consumption records, and coordinating actuator responses in near real time. The laboratory results show that the proposed CM can support these information flows in a structured and computationally feasible way.

This point is relevant because AFS technologies are increasingly associated with improved management precision, lower nutrient oversupply, and better alignment between feed supply and animal requirements. In that context, the present work does not demonstrate direct economic or environmental gains by itself, but it provides part of

the enabling digital infrastructure required for those benefits to be realized in practice. This makes the contribution mainly infrastructural: the paper advances the control and interoperability layer that links decision support outputs with field execution in AFS-based precision feeding.

5.4. Scope and Limitations

Several limitations should be considered when interpreting these findings. First, the validation was conducted in a laboratory setting using a DT and HIL configuration, so the results do not capture the full behavioral, environmental, and network variability of commercial farms. Second, scalability was assessed empirically up to 50 virtual devices, and no direct evidence is provided for larger deployments. Third, the study did not include formal comparison against alternative software environments, message formats, or cybersecurity mechanisms. As a result, the paper supports the feasibility of the implemented solution, but not claims of superiority over Python-, C++-, ROS-, JSON-, or binary-based alternatives. Future work should therefore extend validation to multi-farm conditions, supported operating systems, stronger security mechanisms, and broader performance comparisons.

5.5. Future Work

Future work will focus on aligning the current XML schema with reference ontologies such as SAREF and SAREF4AGRI to support automatic interpretation by external DSSs and farm management platforms [33,34,50]. This integration would preserve the domain-specific semantics by utilizing SAREF properties (e.g., saref:relatesToProperty) to link technical measurements directly to their corresponding biological traits defined by ontologies, thereby ensuring that the biological meaning of the data remains intact within the standardized IoT structure [33,34]. Figure 4 illustrates a possible evolution path from the present ad hoc XML cluster toward a fully standardized, ontology-based message structure.

Despite the sound laboratory-scale validation presented here, several considerations related to the complexity of PLF systems remain. Firstly, while the CM proved stable and robust under continuous operation, the overall accuracy of the precision feeding outcome remains dependent on the precision and reliability of the physical sensors (e.g., RFID accuracy, demand sensor functionality). Hardware vulnerability to harsh farm environments, such as dust and ammonia, poses a significant threat to long-term reliability [2]. Furthermore, technical failures or malfunctions require adequate backup plans to prevent animal welfare issues [3]. Further research is required to independently validate the long-term biological and economic impact of PLF technologies across different animal populations and variable farm conditions [4].

Secondly, the architecture can be enhanced by addressing deployment challenges in rural settings, where internet connectivity might be intermittent or slow. It is estimated that only a small percentage of rural landscapes have adequate cellular coverage [51], representing a primary barrier to installing PLF technologies. Future work should focus on integrating more refined data processing and decision-making capabilities closer to the data source, employing distributed intelligence (fog or edge computing) [52]. For instance, the deployment of lightweight deep learning models, such as those based on vision transformers designed for resource-constrained devices, has proven effective in real-time disease detection using aerial imagery [26,53]. Adopting similar lightweight architectures in feeder controllers would reduce dependency on constant cloud connectivity and minimize latency for mission-critical control actions.

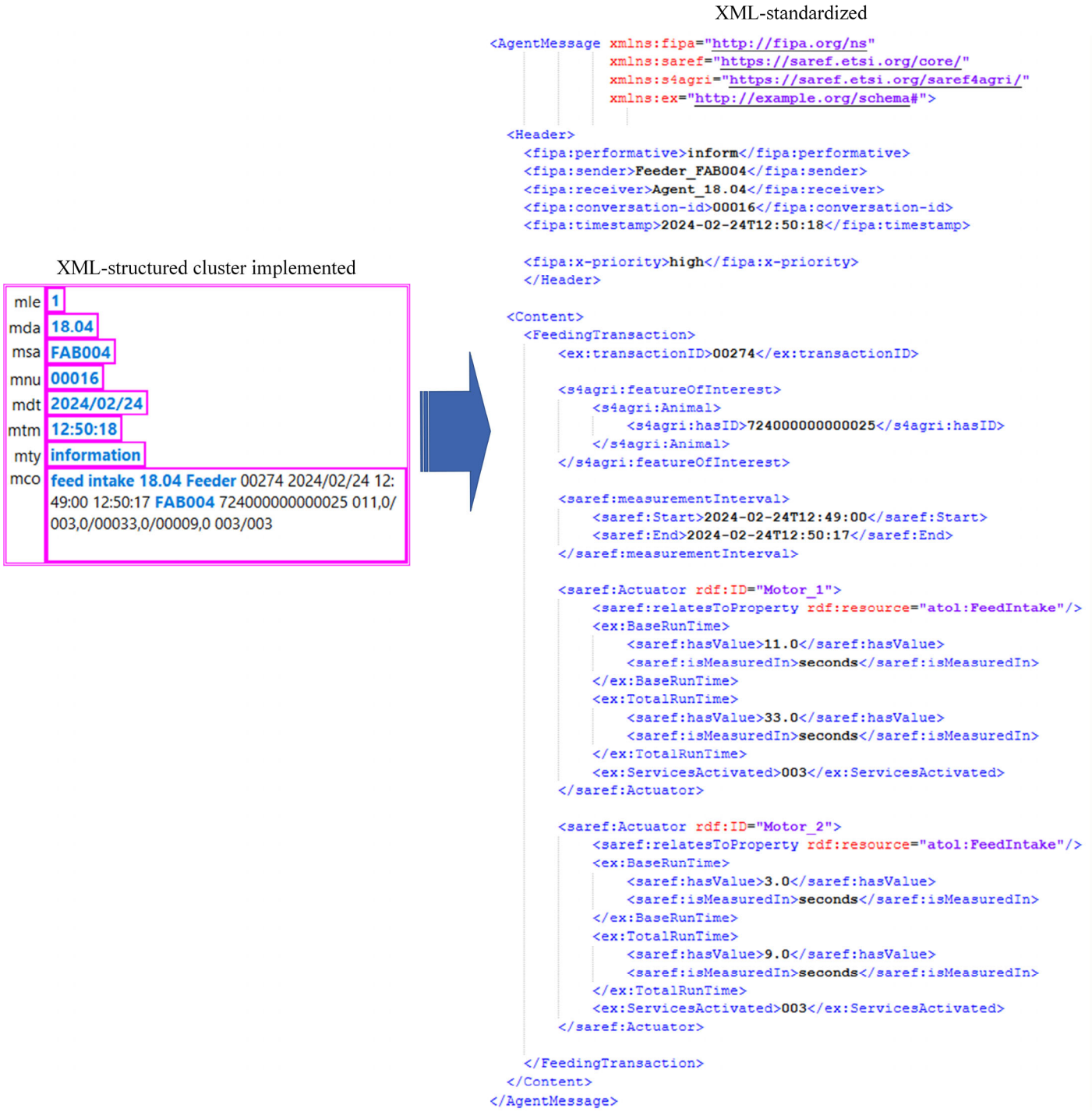


Figure 4. Conceptual evolution of the agent message structure from the current XML-based implementation (**left**) to a standardized, ontology-aligned schema using FIPA ACL headers and SAREF/SAREF4AGRI classes (**right**). This transformation is proposed as future work to enhance semantic interoperability with external systems.

6. Conclusions

This study presented a hybrid MAS architecture for the control of AFSs in precision livestock farming. The proposed CM combined a lightweight character-delimited protocol for communication with field devices and an internal XML-based structure for agent-level message organization. This design was intended to address the practical trade-off between communication efficiency at the edge and structured interoperability within the software layer.

The laboratory validation indicated that the architecture was able to manage sustained communication with 50 virtual devices over 120 h, correctly process the physical sensing events reproduced in the HIL setup, and operate with limited computational resource consumption on constrained hardware. These results support the technical soundness of the proposed approach under the tested conditions. At the same time, they should be interpreted within the scope of a laboratory-scale validation rather than as evidence of full field readiness across diverse production environments.

The main contribution of the work lies in the definition and validation of a control-layer architecture that links DSS outputs with device-level execution in AFS-based precision feeding. In this sense, the study contributes to the digital infrastructure required for individualized feeding strategies, while leaving broader issues such as multi-farm validation, stronger security mechanisms, ontology-level standardization, and comparison with alternative software environments for future research.

Supplementary Materials: The following supporting information can be downloaded at: https://www.mdpi.com/article/10.3390/agriengineering8060242/s1, Figure S1: Block diagram of the FGV Communication.vi.

Author Contributions: Conceptualization, V.L.-S. and L.M.P.-A.; methodology, V.L.-S.; software, V.L.-S.; validation, V.L.-S.; formal analysis, V.L.-S.; investigation, V.L.-S.; writing—original draft preparation, V.L.-S.; writing—review and editing, V.L.-S. and L.M.P.-A.; supervision, L.M.P.-A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by EUROPEAN COMMISSION, grant number 633531 under the EU Framework Programme for Research and Innovation Horizon 2020 and the Generalitat de Catalunya (DARPA) through the Ajuts Demostratius Col·laboratius with the project (2023_SIA002_29) "Menjadores automàtiques i intel·ligència artificial per analitzar el comportament alimentari dels porcs d'engreix i la seva eficiència".

Data Availability Statement: The datasets and software presented in this article are not readily available because they are part of an ongoing study and are currently under active development with planned commercial exploitation. Requests to access the datasets may be considered for academic research and validation purposes under specific conditions.

Acknowledgments: This study was conducted as part of the first author's ongoing PhD thesis. We acknowledge the supervision provided by emeritus Jesús Pomar. Authors also acknowledge the support of CYTED through the thematic network AI4AGROIB Ibero-American network for the development of intelligent systems for agriculture.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ACL	Agent Communication Language
CM	Controlling Module
DSS	Decision Support System
DT	Digital Twin
FIPA	Foundation for Intelligent Physical Agents
FGV	Functional Global Variable
HIL	Hardware-in-the-Loop
IoT	Internet of Things
KIF	Knowledge Interchange Format
KQML	Knowledge Query and Manipulation Language
MAS	Multi-Agent System
PF	Precision Feeding

PLF	Precision Livestock Farming
RFID	Radio Frequency Identification
TCP/IP	Transmission Control Protocol/Internet Protocol
XML	Extensible Markup Language

References

1. Morrone, S.; Dimauro, C.; Gambella, F.; Cappai, M.G. Industry 4.0 and Precision Livestock Farming (PLF): An up to Date Overview across Animal Productions. *Sensors* **2022**, *22*, 4319. [CrossRef]
2. Neethirajan, S.; Kemp, B. Digital Livestock Farming. *Sens. Bio-Sens. Res.* **2021**, *32*, 100408. [CrossRef]
3. Tuyttens, F.A.M.; Molento, C.F.M.; Benaissa, S. Twelve Threats of Precision Livestock Farming (PLF) for Animal Welfare. *Front. Vet. Sci.* **2022**, *9*, 889623. [CrossRef]
4. Gómez, Y.; Stygar, A.H.; M M Boumans, I.J.; M Bokkers, E.A.; Pedersen, L.J.; Niemi, J.K.; Pastell, M.; Manteca, X.; Llonch, P. A Systematic Review on Validated Precision Livestock Farming Technologies for Pig Production and Its Potential to Assess Animal Welfare a Systematic Review on Validated Precision Livestock Farming Technologies for Pig Production and Its Potential to Assess Animal Welfare. *Front. Vet. Sci.* **2021**, *1*, 660565. [CrossRef]
5. Cadéro, A.; Aubry, A.; Brossard, L.; Dourmad, J.Y.; Salaün, Y.; Garcia-Launay, F. Modelling Interactions between Farmer Practices and Fattening Pig Performances with an Individual-Based Model. *Animal* **2018**, *12*, 1277–1286. [CrossRef]
6. Andretta, I.; Pomar, C.; Kipper, M.; Hauschild, L.; Rivest, J. Feeding Behavior of Growing–Finishing Pigs Reared under Precision Feeding Strategies1. *J. Anim. Sci.* **2016**, *94*, 3042–3050. [CrossRef]
7. Andretta, I.; Hauschild, L.; Kipper, M.; Pires, P.G.S.; Pomar, C. Environmental Impacts of Precision Feeding Programs Applied in Pig Production. *Animal* **2018**, *12*, 1990–1998. [CrossRef]
8. Gaillard, C.; Dourmad, J.-Y. Application of a Precision Feeding Strategy for Gestating Sows. *Anim. Feed Sci. Technol.* **2022**, *287*, 115280. [CrossRef]
9. Pomar, C.; Milgen, J.v.; Remus, A. 18: Precision Livestock Feeding, Principle and Practice. In *Poultry and Pig Nutrition*; Wageningen Academic: Wageningen, The Netherlands, 2019; pp. 397–418. [CrossRef]
10. Su, J.; Tan, B.; Jiang, Z.; Wu, D.; Nyachoti, C.M.; Kim, S.W.; Yin, Y.; Wang, J. Accelerating Precision Feeding with the Internet of Things for Livestock: From Concept to Implementation. *Sci. Bull.* **2024**, *69*, 2156–2160. [CrossRef]
11. Bao, Y.; Llagostera, P.; Plà-Aragonès, L.M. Is Deep Learning Useful for Decision Making in Pig Production? *Internet Things* **2024**, *26*, 101229. [CrossRef]
12. Riahi Sfar, A.; Natalizio, E.; Challal, Y.; Chtourou, Z. A Roadmap for Security Challenges in the Internet of Things. *Digit. Commun. Netw.* **2018**, *4*, 118–137. [CrossRef]
13. Arulmozhi, E.; Bhujel, A.; Moon, B.-E.; Kim, H.-T. The Application of Cameras in Precision Pig Farming: An Overview for Swine-Keeping Professionals. *Animals* **2021**, *11*, 2343. [CrossRef]
14. Kaur, U.; Malacco, V.M.R.; Bai, H.; Price, T.P.; Datta, A.; Xin, L.; Sen, S.; Nawrocki, R.A.; Chiu, G.; Sundaram, S.; et al. Invited Review: Integration of Technologies and Systems for Precision Animal Agriculture—A Case Study on Precision Dairy Farming. *J. Anim. Sci.* **2023**, *101*, skad206. [CrossRef]
15. Friha, O.; Ferrag, M.A.; Shu, L.; Maglaras, L.; Wang, X. Internet of Things for the Future of Smart Agriculture: A Comprehensive Survey of Emerging Technologies. *IEEE/CAA J. Autom. Sin.* **2021**, *8*, 718–752. [CrossRef]
16. Miller, T.; Mikiciuk, G.; Durlik, I.; Mikiciuk, M.; Łobodzińska, A.; Śnieg, M. The IoT and AI in Agriculture: The Time Is Now—A Systematic Review of Smart Sensing Technologies. *Sensors* **2025**, *25*, 3583. [CrossRef]
17. Rustemi, A.; Dalipi, F. Synergizing IoT, AI, and Blockchain for Smart Agriculture: Challenges, Opportunities, and Future Directions. *Internet Things* **2025**, *34*, 101778. [CrossRef]
18. Yu, P.; Teng, F.; Zhu, W.; Shen, C.; Chen, Z.; Song, J. Cloud–Edge–Device Collaborative Computing in Smart Agriculture: Architectures, Applications, and Future Perspectives. *Front. Plant Sci.* **2025**, *16*, 1668545. [CrossRef]
19. Singh, R.; Gill, S.S. Edge AI: A Survey. *Internet Things Cyber-Phys. Syst.* **2023**, *3*, 71–92. [CrossRef]
20. Savaglio, C.; Mazzei, P.; Fortino, G. Edge Intelligence for Industrial IoT: Opportunities and Limitations. *Procedia Comput. Sci.* **2024**, *232*, 397–405. [CrossRef]
21. Altin, N.; Eyimaya, S.E.; Nasiri, A. Multi-Agent-Based Controller for Microgrids: An Overview and Case Study. *Energies* **2023**, *16*, 2445. [CrossRef]
22. Barriuso, A.; Villarrubia González, G.; De Paz, J.; Lozano, Á.; Bajo, J. Combination of Multi-Agent Systems and Wireless Sensor Networks for the Monitoring of Cattle. *Sensors* **2018**, *18*, 108. [CrossRef]
23. *SC00061G*; FIPA ACL Message Structure Specification. FIPA TC Communication. FIPA: Geneva, Switzerland, 2002; p. 11. Available online: https://web.archive.org/web/20250130105414/http://www.fipa.org/specs/fipa00061/SC00061G.pdf (accessed on 10 February 2026).
24. Collins, L.M.; Smith, L.M. Review: Smart Agri-Systems for the Pig Industry. *Animal* **2022**, *16*, 100518. [CrossRef]

25. Marić, K.; Gvozdanović, K.; Djurkin Kušec, I.; Kušec, G.; Margeta, V. Smart Pig Farms: Integration and Application of Digital Technologies in Pig Production. *Agriculture* **2025**, *15*, 937. [CrossRef]
26. Savaglio, C.; Barbuto, V.; Mangione, F.; Fortino, G. Generative Digital Twins: A Novel Approach in the IoT Edge-Cloud Continuum. *IEEE Internet Things Mag.* **2025**, *8*, 42–48. [CrossRef]
27. Pomar, C.; Pomar, J.; Rivest, J.; Cloutier, L.; Letourneau-Montminy, M.P.; Andretta, I.; Hauschild, L. Estimating Real-Time Individual Amino Acid Requirements in Growing-Finishing Pigs: Towards a New Definition of Nutrient Requirements in Growing-Finishing Pigs? *Nutr. Model Pigs Poult.* **2015**, 157–174. [CrossRef]
28. Pomar, C.; Remus, A. Precision pig feeding: A breakthrough toward sustainability. *Anim. Front.* **2019**, *9*, 52–59. [CrossRef] [PubMed]
29. Andretta, I.; Pomar, C.; Rivest, J.; Pomar, J.; Lovatto, P.A.; Radünz Neto, J. The impact of feeding growing-finishing pigs with daily tailored diets using precision feeding techniques on animal performance, nutrient utilization, and body and carcass composition. *J. Anim. Sci.* **2014**, *92*, 3925–3936. [CrossRef]
30. Llorens, B.; Pomar, C.; Goyette, B.; Rajagopal, R.; Andretta, I.; Latorre, M.A.; Remus, A. Precision feeding as a tool to reduce the environmental footprint of pig production systems: A life-cycle assessment. *J. Anim. Sci.* **2024**, *102*, skae225. [CrossRef]
31. Castillejo, P.; Cürüklü, B.; Fresco, R.; Johansen, G.; Bilbao-Arechabala, S.; Martinez-Rodriguez, B.; Pomante, L.; Martinez-Ortega, J.-F.; Santic, M. The AFarCloud ECSEL Project. In *Proceedings of the 2019 22nd Euromicro Conference on Digital System Design (DSD), Kallithea, Greece*; IEEE: New York, NY, USA, 2019; pp. 414–419. [CrossRef]
32. ETSI. SmartM2M; Extension to SAREF; Part 6: Smart Agriculture and Food Chain Domain (SAREF4AGRI), ETSI TS 103 410-6 V2.1.1, 2024. Available online: https://www.etsi.org/deliver/etsi_ts/103400_103499/10341006/02.01.01_60/ts_10341006v020101p.pdf (accessed on 10 February 2026).
33. García, R.; López-Gil, J.-M.; Gil, R. Rhizomer: Interactive semantic knowledge graphs exploration. *SoftwareX* **2022**, *20*, 101235. [CrossRef]
34. Li, Y.; Jiayuan, W.; Zhouli, Y.; Haiyu, Z. Agent Technology for Agricultural Intelligence: Methodological Framework and Applications. *Electronics* **2026**, *15*, 1547. [CrossRef]
35. Pomar, J.; López, V.; Pomar, C. Agent-Based Simulation Framework for Virtual Prototyping of Advanced Livestock Precision Feeding Systems. *Comput. Electron. Agric.* **2011**, *78*, 88–97. [CrossRef]
36. Polaków, G.; Metzger, M. Agent–Based Approach for LabVIEW Developed Distributed Control Systems. In *Agent and Multi-Agent Systems: Technologies and Applications*; Nguyen, N.T., Grzech, A., Howlett, R.J., Jain, L.C., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4496, pp. 21–30. [CrossRef]
37. Zimmermann, O. Comprendre la FGV et Ses Extensions Dans LabVIEW, 2018. Available online: https://lpsc-indico.in2p3.fr/event/1125/attachments/274/359/180323_ZIMMERMANN_Olivier.pdf (accessed on 10 February 2026).
38. National Instruments. Functional Global Variable (FGV). Available online: https://learn-cf.ni.com/teach/riodevguide/code/rt_functional-global-variable.html (accessed on 10 February 2026).
39. Kodosky, J. LabVIEW. *Proc. ACM Program. Lang.* **2020**, *4*, 1–54. [CrossRef]
40. Maturana, F.; Staron, R.; Discenzo, F.; Hall, K.; Tichý, P.; Šlechta, P.; Mařík, V.; Scheidt, D.; Pekala, M.; Bracy, J. Integrating Multi-Agent Systems: A Case Study. In *Emerging Solutions for Future Manufacturing Systems*; Camarinha-Matos, L.M., Ed.; IFIP International Federation for Information Processing; Kluwer Academic Publishers: Boston, MA, USA, 2005; Volume 159, pp. 99–108. [CrossRef]
41. Reddy, K.V.; Reddy, B.S.K.; Goutham, V.; Mahesh, M.; Nisha, J.S.; Palanisamy, G.; Golla, M.; Purushothaman, S.; Reddy, K.R.; Ramkumar, V. Edge AI in Sustainable Farming: Deep Learning-Driven IoT Framework to Safeguard Crops from Wildlife Threats. *IEEE Access* **2024**, *12*, 77707–77723. [CrossRef]
42. Wolfert, S.; Ge, L.; Verdouw, C.; Bogaardt, M.-J. Big Data in Smart Farming—A Review. *Agric. Syst.* **2017**, *153*, 69–80. [CrossRef]
43. Vedantham, A.K. Cloud and IoT Technologies Revolutionizing Precision Agriculture. *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol.* **2024**, *10*, 860–867. [CrossRef]
44. Leitão, P. Agent-Based Distributed Manufacturing Control: A State-of-the-Art Survey. *Eng. Appl. Artif. Intell.* **2009**, *22*, 979–991. [CrossRef]
45. Villafranca, A.; Cano, M.-D. A Lightweight Edge-DL Intrusion Detection System for IoT Sustainable Smart-Agriculture. *Internet Things* **2025**, *34*, 101818. [CrossRef]
46. Loaiza, M.; Savaglio, C.; Arijo, N.H.; Aloi, G.; Fortino, G.; Gravina, R. Agents in Software Development Architectures. In Proceedings of the WOA 2023: 24th Workshop from Objects to Agents, Rome, Italy, 6–8 November 2023. CEUR Workshop Proceedings.
47. Genesereth, M.R.; Fikes, R.E. Knowledge Interchange Format Version 3.0 Reference Manual, 1992. Available online: http://logic.stanford.edu/publications/genesereth/kif.pdf (accessed on 10 February 2026).
48. Pěchouček, M.; Mařík, V. Industrial Deployment of Multi-Agent Technologies: Review and Selected Case Studies. *Auton. Agent. Multi-Agent. Syst.* **2008**, *17*, 397–431. [CrossRef]
49. Genesereth, M.R.; Ketchpel, S.P. Software Agents. *Commun. ACM* **1994**, *37*, 48–53. [CrossRef]

50. Tomic, D.; Drenjanac, D.; Hoermann, S.; Auer, W. Experiences with Creating a Precision Dairy Farming Ontology (DFO) and a Knowledge Graph for the Data Integration Platform in agriOpenLink. *J. Agric. Inform.* **2015**, *6*. [CrossRef]
51. Cantera, J.M.; Issa, J.S.; van der Vlugt, P.; Klaeser, S.; Bartram, T.; Kassahun, A.; Neira, I.; Milin, T. D3.3 Opportunities and Barriers in the Present Regulatory Situation for System Development; Internet of Food and Farm 2020; Deliverable D3.3; 2018; p. 93. Available online: https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5bb2a8e03&appId=PPGMS (accessed on 10 February 2026).
52. Kakamoukas, G.; Sarigiannidis, P.; Maropoulos, A.; Lagkas, T.; Zaralis, K.; Karaiskou, C. Towards Climate Smart Farming—A Reference Architecture for Integrated Farming Systems. *Telecom* **2021**, *2*, 52–74. [CrossRef]
53. Li, Q.; Chen, W. Lightweight Vision Transformer Model for Pine Wilt Disease Detection Using Aerial RGB Image and Adversarial Data Augmentation. *Smart Agric. Technol.* **2025**, *12*, 101167. [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.