Flexible, scalable, and robust architecture for industrial automation applications with real-time requirements

Sergio Luis Castaño Rodriguez
Facultad de Ingeniería
Universidad Autónoma de Occidente
Cali, Colombia
slcastano@uao.edu.co

Diego Martinez Castro
Facultad de Ingeniería
Universidad Autónoma de Occidente
Cali, Colombia
dmartinez@uao.edu.co

***Abstract*— With the outbreak of the fourth industrial revolution, industrial automation applications demand a more flexible, scalable, and robust architecture than the one proposed in the ISA-95 standard. In addition, the high availability demanded by industrial processes hinders maintenance and updating tasks that go against information security policies. To address these challenges, some frameworks supported by virtualization and information distribution technologies have been generated for industrial automation systems. However, these proposals lack the means to implement real-time task scheduling algorithms, so it is difficult to migrate developments already implemented under these requirements. In this work, we present a microservices-based architecture that integrates virtualization technologies, a robust middleware for the communication layer, and a priority-based task scheduler. It also describes the implementation of this architecture using low-cost hardware components and free software, to be subsequently tested on a pressure control process in a laboratory setting. The obtained results confirm the effectiveness of the proposed architecture and meeting of real-time requirements, which enables supporting the diverse current industrial automation applications requirements, addressing the challenges of migration to Industry 4.0, and incorporating new functionalities.**

Keywords—Industrial automation, real-time, virtualization, interoperability, middleware, Industry 4.0

I. INTRODUCTION

The ISA-95 standard was developed to allow efficient integration of operations and manufacturing in industrial automation applications [1]. It facilitates the description of models and terminology for determining information exchange between different levels of the system. It defines different automation levels, ranging from physical processes at the lowest level to the activities related to the manufacturing business at the highest levels.

Currently, the industry is seeking to expand the automation of its processes demanding flexibility and scalability, which also represents an increase in the number and resources of hardware nodes. As an alternative, technologies such as virtualization make possible the inclusion of new components at low costs, contributing in terms of flexibility and information exchange between components. Moreover, virtualization provides resource efficiency, easier management, minimal downtime, and faster provisioning. Exploration of the benefits promised by virtualization in industrial automation systems is motivated by several key aspects. For instance, a large number of interconnected hardware units are costly and error-prone but exhibit strong independence; virtualization technologies provide independence and cost-effective hardware at the same time. Besides, integration of new solutions becomes necessary when legacy hardware is no longer available due to the long lifetime of plant facilities and virtualization can help in this regard [1].

As with traditional approaches, the implementation of virtualization in industrial applications entails verifying compliance with real-time requirements. Some virtualization architectures provide temporal isolation, which means applications run independently and the execution time of one does not affect the others. This prevents a virtual partition from affecting the ability of others to access a shared resource [2], thus meeting application deadlines.

Containers and virtual machines are two virtualization mechanisms that provide comparable benefits in terms of resource allocation and isolation. However, containers are considered more portable and efficient due to their architecture, requiring fewer resources. This facilitates the standardization, sharing, development, deployment, and security of applications and modules. The objective of containers is to encapsulate simple microservices that operate on a network, providing a modular approach to application construction [3].

Microservices-based software architectures have recently been used to solve a wide variety of complex problems, that must adapt to changing demands. A microservice is a small, single-responsibility application that can be independently deployed, scaled, and tested. Microservices can replace monolithic architectures by enabling the development of distributed, lightweight, decoupled, and independent service applications that work together. In [4], an architecture is presented that incorporates cloud computing for data processing in agricultural applications that do not require meeting real-time. The non-deterministic behavior of the communication network does not affect application performance in such cases. However, real-time applications cannot guarantee meeting deadlines using this architecture.

Since microservices can be implemented separately in different hardware nodes or processes, there may be a network communication overhead between them that affects the performance of distributed systems. Therefore, an efficient network communication protocol is required.

In this paper, we propose a new microservice-based architecture for industrial automation systems. This architecture is characterized by the simultaneous integration of microservices with real-time execution policies and containerized non-real-time microservices. Real-time policy support is achieved through a scheduler capable of handling tasks with fixed priorities, provided by Linux patched with PREEMPT_RT. To ensure reliable and robust inter-container communication, the Data Distribution Service (DDS) is used, which is a standard data-centric middleware based on the publisher/subscriber paradigm. With the integration of these techniques, this architecture allows the development of flexible, scalable, and robust systems, as well as the fulfillment of real-time requirements, thus satisfying the previous and current needs of industrial automation applications.

II. Current Status of Virtualization Technologies in Industrial Automation

Although various virtualization solutions are commercially available, both server (e.g., ESXi, XenServer) and real-time embedded (e.g., Integrity Multivisor, Wind River Hypervisor), the use of virtualization in the industrial automation sector is relatively new. Therefore, there are challenges and opportunities to integrate virtualization technologies into industrial processes [1, 5], such as legacy support, smart sensors and actuators, increased connectivity, network security, predictability of network latencies, shift towards commodity hardware, mixed-technologies, mixed-criticality, and legacy hardware emulation.

A study [6] explored the challenges of applying virtualization techniques in real-time embedded systems. The use of a hypervisor-based virtualization solution presents challenges when adapting it to industrial automation legacy systems due to the overhead. It may be possible to overcome this issue by leveraging instruction set emulation and advanced processor generations [7]. Moreover, hypervisor-based virtualization exhibits restricted granularity in the encapsulation of the functionality [8], resulting in the inclusion of legacy functionality at the application level [9] rather than utilizing virtualization methods. However, the literature on virtualization for programmable logic controllers (PLCs) and similar industrial devices tends to focus on hypervisor-based techniques.

Another significant issue of implementing virtualization on industrial automation devices is guaranteeing compliance with execution deadlines using available virtualization solutions. In [10], real-time containers were utilized to process real-time data and meet strict deadlines for event detection and response. The paper emphasizes the importance of virtualization at the operating system level, as well as synchronization between containers in the context of industrial automation systems. The effects of containers on the performance of industrial automation systems were evaluated in two cases: the cyclic behavior of a containerized application and the performance of the virtual network for inter-container communications. The cyclic behavior test assesses the capacity to perform the application logic at specified intervals, measuring accuracy and jitter. The virtual network test evaluates communication among co-located containers within a time limit. The studies concluded real-time container computing is a promising technology, but effective strategies for inter-container communication require further investigation.

In [11], a real-time container-based architecture for industrial controllers was presented to enable flexible feature implementation and support of legacy control applications. Such an architecture adds value by preserving the functionality of legacy control programs and reducing the cost of maintaining legacy systems (where software and hardware are limited to a specific ecosystem). A series of tests were performed on two scenarios, the first using applications wrapped in Docker containers, and the second emulating a full legacy operating system (QEMU PowerPC emulator) inside Linux Containers (LXC). Docker and LXC are common implementations of tools and APIs for creating and managing containers on Linux. The work concludes that the execution of containerized control applications using the proposed architecture can meet the real-time requirements of PLCs and automation controllers with cycle times above 100ms.

The aforementioned works meet real-time demands by employing containers on an operating system equipped with real-time capabilities, specifically Linux with the real-time patch (PREEMPT_RT). However, the use of two kernels, a real-time microkernel that runs in parallel with the Linux kernel, is another approach. The real-time co-kernel manages time-critical tasks, while the standard Linux kernel runs only when the co-kernel is idle. Compared to a single kernel (real-time patch), the co-kernel approach provides lower latencies and reduced jitter. On the other hand, this approach requires special APIs, tools, and libraries for application development. Additionally, there are obstacles to scaling co-kernel solutions on large platforms (e.g., many cores platforms). Real-Time Application Interface (RTAI) and Xenomai are both commonly used options for co-kernels.

A modular architecture for real-time control applications using real-time containers supported by a co-kernel approach (combination of Xenomai and real-time patch) was presented in [12]. The exchange of messages between real-time containers is one of the major obstacles in this type of architecture. The containers cannot distinguish whether they operate on the same host or in a distributed environment, therefore direct message passing through shared memory is not directly supported. In response, the researchers designed and implemented a customized real-time messaging system for containers based on ZeroMQ. In [13], real-time containers were implemented utilizing the real-time co-kernel (RTAI), custom monitoring, and policy enforcing modules. This work provides temporal guarantees through two mechanisms: assigning appropriate priority levels to tasks inside containers and executing temporal protection policies while constant monitoring. The former ensures that tasks in high-criticality containers are given higher priority than tasks in low-criticality containers, and therefore are never preempted by tasks in containers with lower criticality. The latter monitors the tasks and, in cases of overrun or overtime, applies one of the temporary protection policies, such as suspending the task until the next period.

In a separate approach to kernel intervention, a system supported by a custom scheduling mechanism for real-time and non-real-time containers has been proposed in [14]. The scheduler guarantees that real-time containers receive the allocated CPU capacity while non-real-time containers receive the unutilized capacity distributed dynamically. The research enhances the Completely Fair Scheduler (CFS) by introducing a workload adjustment module that gathers the CPU utilization of containers and a dynamic adjustment module that assigns CPU resources to the container.

Complying with time constraints in distributed control systems presents a significant challenge due to the execution of various tasks in separate processing systems, as well as the need to consider network resources. In [15], a framework was presented in order to utilize containerized microservices components (using Singularity container technology) to enable simple implementation, scalability, and fast maintenance without compromising the robustness of process control applications. The researchers used Linux cgroups to perform local resource scheduling, which is a tool used to manage resource allocation to processes. By allocation of containers to the CPU, the usage percentage can be balanced. Data Distribution Service (DDS) middleware was implemented to overcome the communication issue of time-sensitive distributed applications. The DDS middleware provides control by adjusting the behavior of network traffic through the quality of service (QoS) settings.

DDS has demonstrated its effectiveness as a middleware in real-time systems. An instance of this was presented in [16], where it was utilized to exchange data and incorporate multiple components of an Adaptive Cruise Control (ACC) system. The outcomes indicate that DDS is a viable and practical solution, reporting latencies slightly below 100 milliseconds, qualifying it as appropriate for the development of certain real-time systems. Additionally, strategies have been developed to improve the reliability and performance of communications in DDS-based systems. In [17], a novel algorithm was proposed to adapt the transmission rate, optimizing the reliability and performance of message delivery. Experimental results showed that the proposed algorithm achieves over 99% system communication reliability.

III. Node Architecture for Industry 4.0

Fig. 1 illustrates the proposed architecture for achieving the flexibility, scalability, robustness, and meeting real-time deadlines required for industry 4.0. The main feature of this architecture is the integration of a real-time kernel, supported by the Linux PREEMPT_RT patch [18], which allows the modeling and validation of real-time applications. This is intended to ensure support for control applications that must meet strict deadlines. The advantage of using the Linux kernel to satisfy real-time applications over other alternatives with dual kernel schemes, such as RTAI and Xenomai, is that it requires less effort to perform maintenance, updating, and adaptation to hardware, which favors flexibility.

In the context of implementing this architecture, we propose a systematic segregation of industrial automation software components into two distinct categories: non-real-time microservices and real-time tasks. Microservices responsible for performing non-real-time functions, such as report generation, data capture from auxiliary sensors, graphical user interface handling, database operations, etc., are enclosed in containers and executed as traditional operating system processes. The adoption of containerization offers notable advantages, including enhanced scalability, portability, and streamlined maintenance of applications, critical not only for updates but also for the implementation of security strategies [19]. On the other hand, functions necessitating stringent adherence to real-time deadlines should be implemented as fixed high-priority tasks grouped into a single process. This approach, in contrast to containerized implementations, results in reduced utilization of memory and CPU.

Using containers that operate efficiently in parallel with real-time tasks requires prioritizing performance as the primary selection criterion for the container engine. This guarantees that process execution times are comparable to native execution and that the overall system performance is optimized. Similarly, it is essential to assess the network performance of containers since this architecture was designed to enable hardware node interconnection in a distributed automation system. Therefore, it was decided to use Singularity as the container engine for implementing the proposed architecture. It has shown superior performance in demanding computing tasks in the context of high-performance computing (HPC) [20] and nearly native results in terms of computational and network performance [21].

For communication among microservices in containers, the DDS standard middleware was selected. DDS participants can be located on the same machine or across a network. One significant advantage of this protocol, as opposed to other data-centric communication protocols like MQTT, is its decentralized communication model, as it allows participants to communicate peer-to-peer without the need for a server to broker the data. Decentralization enhances system robustness by eliminating a single point of failure. In addition, by implementing DDS technology, microservices do not need to configure communication endpoints as they are automatically

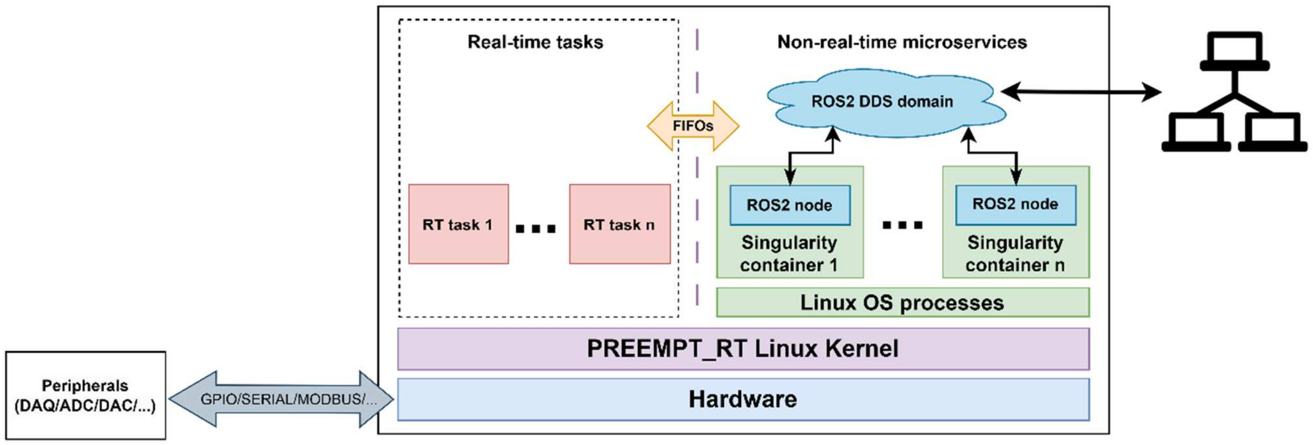

Fig. 1. Architecture proposed for nodes in industrial automation systems

discovered. This allows for the addition, termination, and removal of microservices without affecting the entirety of the application. [22]

The second version of the Robot Operating System (ROS2) is a framework that incorporates DDS as the data transport layer, which favors the development of applications based on microservices. In this framework, the DDS participants are called "nodes" and are software pieces where data is processed. Nodes exchange data through "topics" using the publisher/subscriber scheme.

Thus, we propose the development of containerized ROS2 nodes for each microservice that does not require real-time. This approach establishes a shared middleware between the containers using DDS, which enhances reliability and scalability while providing low-latency connectivity and modularity. Data exchange between real-time tasks and containerized microservices occurs through named pipes (FIFOs), while access to field devices utilizes connection ports and platform-specific protocols.

IV. Case Study

The pressure control process shown in Fig. 2 was used as a study case to validate the proposed architecture. The actuator of the system is a valve that regulates the inlet flow to the tank by changing its opening ratio through voltage levels. A relative pressure transmitter is used to measure the variable to be controlled. To represent the air demand in the process, a fixed partial opening valve is installed at the outlet of the tank. In the proposed scenario, setpoint selection and pressure monitoring are performed by a Human-Machine Interface (HMI) running on a device remote from the plant.

The control strategy implemented was a PID control algorithm running in a node (controller node) with the proposed architecture. For the controller node implementation, the criteria of low-cost hardware were taken, selecting a Raspberry Pi 4 (RPI4), which is an embedded system already implemented in some industrial PLCs [23]. The Linux distribution used is the Raspberry Pi OS patched with PREEMPT_RT. Since the RPI4 does not have the hardware to directly interface with the plant, an Arduino Due was used as the analog signal interface. RPI4 and Arduino Due communicate through the serial port. The remote HMI was executed on a PC running Ubuntu 22.04. RPI4

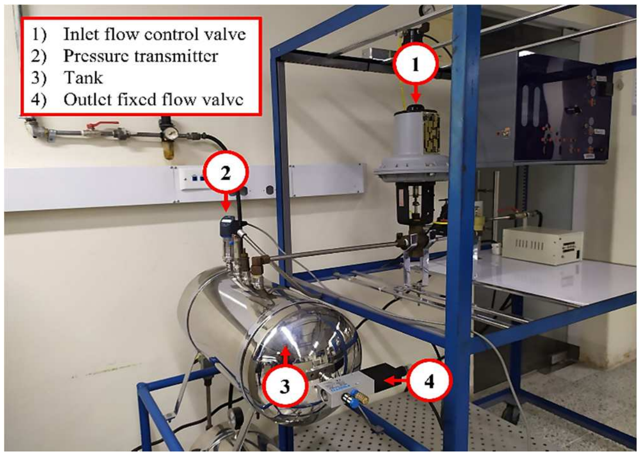

Fig. 2. Illustration of the plant used as a case study for the validation of the architecture proposed.

and PC exchange data via DDS middleware over a LAN network.

Fig. 3 shows the distribution of the software components implemented in the application under the proposed architecture, which are described below.

- *Container 1* – Singularity container employed to execute the ROS2 middleware where Node A is instantiated. Node A is configured to subscribe to the Setpoint topic. Whenever a message is published on Setpoint topic, Node A captures the data and subsequently writes it to a designated named pipe (fifo_ROS_to_RT), serving as the communication conduit between the ROS2 middleware and the real-time task.
- *Container 2* – Singularity container employed to execute the ROS2 middleware where Node B is instantiated. Node B operates in response to newly written data by the real-time task 3 into the designated named pipe (fifo_RT_to_ROS). Upon detecting new data, Node B reads and encapsulates it within a message, subsequently publishes it onto the Sensor topic.
- *Container 3* – This microservice was implemented on the remote PC and corresponds to a Singularity container running the ROS2 middleware where the "rosbridge_server" node is executed. This particular node is a component of a package developed by ROS community [24] and was designed to establish a transport layer through WebSockets, allowing interfacing to the ROS2 domain from a web browser.
- *Browser* – Representation of the Web HMI interface (Fig. 4) which allows interaction with ROS2 domain topics through WebSockets provided by rosbridge_server. It subscribes to the Sensor topic to retrieve pressure data and display it graphically. Furthermore, the interface publishes to the Setpoint topic, allowing the user to transmit the selected pressure level reference value.
- *Real-time Task 1*- Updates the setpoint value select by the user in the web HMI. Task 1 operates in a continuous monitoring mode, observing a designated named pipe (FIFO ROS to RT) in anticipation of new data to be written by Node A.
- *Real-time Task 2* - Executes the control loop. Initially, it sends a request to the analog interface to obtain a sensor reading. Upon receiving the updated pressure value, it proceeds to calculate the control action and subsequently transmits it to the analog interface to have an effect in the plant. After that, waits until next sample time.
- *Real-time Task 3* – Writes the data corresponding to the last pressure reading made during Task 2 execution on a periodic basis, at a 5 Hz rate, into the designated named pipe (FIFO RT to ROS) for reception by Node B.

A linear approximation of the plant model within a 20 PSI to 30 PSI operational range was utilized to computed the control algorithm parameters to yield an overdamped response with 30 second settling time. The HMI is illustrated in Fig. 4, wherein a time evolution graph portrays the performance of the control system. The observed response aligns closely with the stipulated behavior by the theoretical design, particularly in the middle part of the operational range.

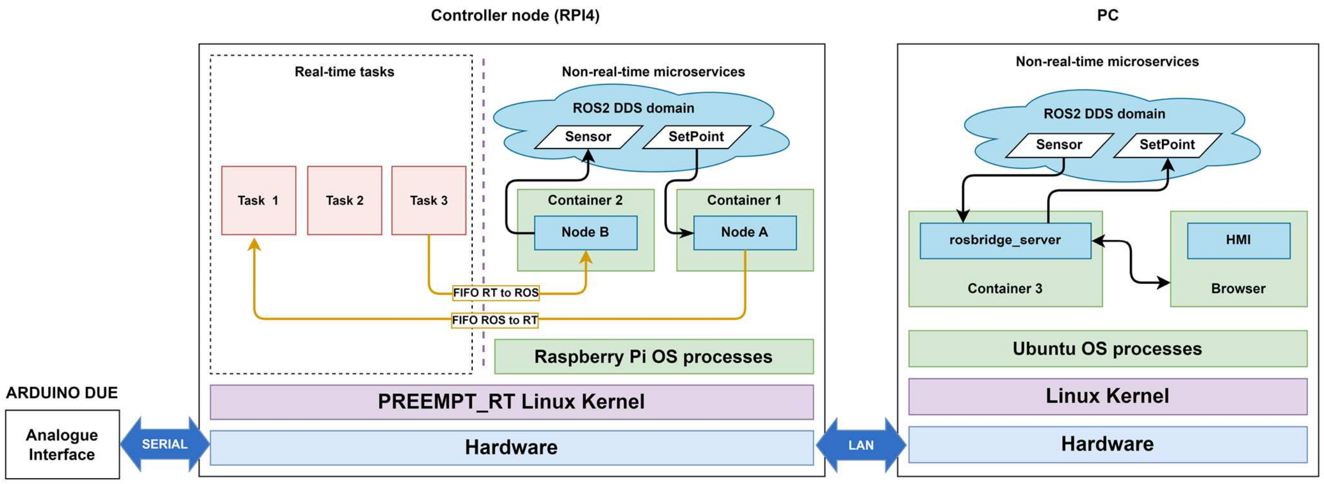

Fig. 3. Components of the application implemented for the case study based on the proposed architecture.

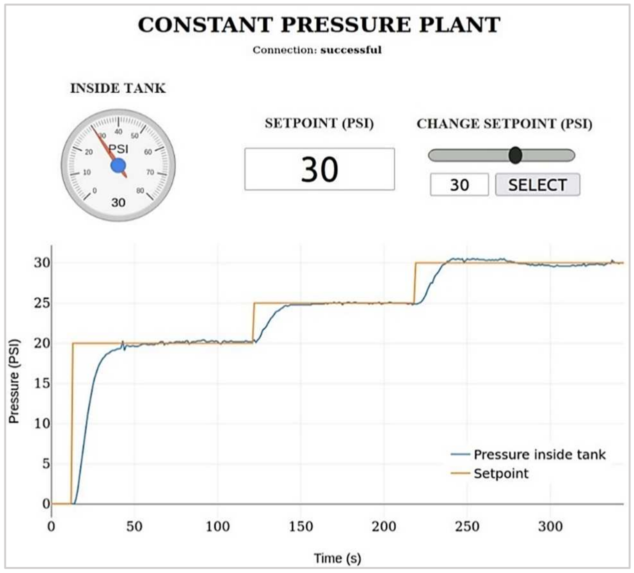

Fig. 4. Web HMI of the application

To achieve the intended dynamic response from the system, the control cycle must be executed precisely every 100ms, which includes measurement of the variable, calculation, and application of the control signal. The effectiveness in accurately meet this period was evaluated in three scenarios, where three thousand (3000) measurements of the thread activation period related to the execution of the control loop were captured (Real-time Task 2).

The first scenario (test 1) was the control scenario, in which the only tasks with real-time scheduling policies were those related to the control loop (Real-time Task 1, 2, and 3). In a second scenario (test 2), two extra cyclic tasks with real-time scheduling policies were executed, each with an execution period of 50ms, one with a fixed priority higher and the other with a fixed priority lower than the control loop tasks. In the third scenario (test 3), both extra tasks were assigned with higher priorities than the control loop tasks, meaning that the scheduler will be able to preempt those tasks.

The distribution of the measured data for each scenario is presented in Fig. 5 and selected measures of central tendency and dispersion of the time parameters obtained are presented in Table I.

The effect that including other tasks with higher execution priorities has on data dispersion is evident. The competition for resources and the possibility of being preempted by other tasks directly affects processes such as the one in charge of calculating the thread waiting time to satisfy the periodicity, which is

TABLE I
PERIOD MEASUREMENT RESULTS

Scenario	*Mean*	*Median*	*Standard deviation*	*Maximum*	*Minimum*
Test 1	99.999997	100.0	0.005606	100.053	99.9627
Test 2	99.997817	99.9977	0.007416	100.060	99.9533
Test 3	99.997590	99.9974	0.011039	100.066	99.9502

All units in milliseconds (ms)

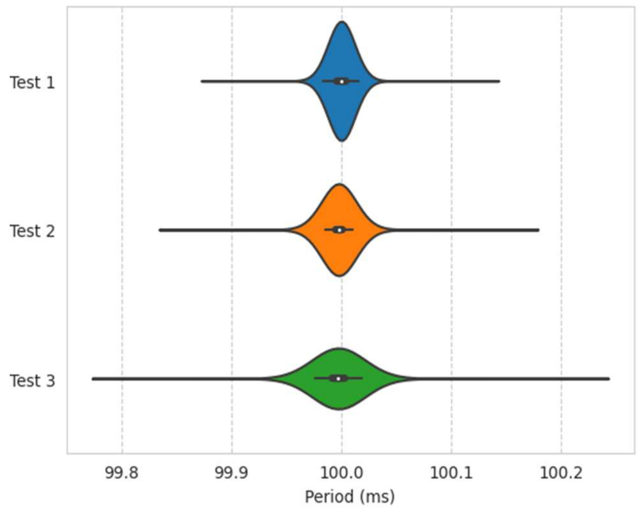

Fig. 5. Measured data distribution.

determined in each iteration to compensate the variation in execution times that may occur in the control loop without affecting the period. However, the range of time measurements shows a low jitter in every scenario, thus, based on the experimental results presented, it is possible to conclude that the proposed architecture allows to fulfill with real time requirements in the node, while presenting a flexible environment to develop applications.

V. Conclusions

The new industrial automation applications demand great flexibility, this added to the inherited support of the current applications in relation to the availability demanded by the industrial processes, raise the need for new architectures in the industrial automation processes.

This study introduces a novel architecture designed for the nodes constituting industrial automation applications. By integrating containerized microservices that communicate through DDS middleware and real-time tasks supported by a Linux distribution enhanced with the PREEMPT_RT patch, this architecture achieves a flexible, scalable and robust infrastructure. Furthermore, it provides the necessary capabilities to meet stringent real-time constraints.

To demonstrate the viability of the proposed architecture, a case study was conducted in an air pressure control process. The obtained results substantiate that the implemented solution effectively fulfills the objectives outlined for the application. This achievement not only aligns with the historical requirements of industrial automation applications but also caters to emerging needs, thereby streamlining design processes through the seamless interconnection of easily adaptable components.

References

[1] H. P. Breivold, A. Jansen, K. Sandström and I. Crnkovic, "Virtualize for Architecture Sustainability in Industrial Automation," 2013 IEEE 16th International Conference on Computational Science and Engineering, Sydney, NSW, Australia, 2013, pp. 409-415, doi: 10.1109/CSE.2013.69.

[2] A. Crespo, I. Ripoll and M. Masmano, "Partitioned Embedded Architecture Based on Hypervisor: The XtratuM Approach," 2010 European Dependable Computing Conference, Valencia, Spain, 2010, pp. 67-72, doi: 10.1109/EDCC.2010.18.

[3] P. González-Nalda, I. Etxeberria-Agiriano, I. Calvo, and M. C. Otero, "A modular CPS architecture design based on ROS and Docker," International Journal on Interactive Design and Manufacturing (Ijidem), vol. 11, no. 4, pp. 949–955, Apr. 2016, doi: 10.1007/s12008-016-0313-8.

[4] R. Chandra et al., "Democratizing Data-Driven Agriculture Using Affordable Hardware," in IEEE Micro, vol. 42, no. 1, pp. 69-77, 1 Jan.-Feb. 2022, doi: 10.1109/MM.2021.3134743.

[5] V. Struhar, M. Behnam, M. Ashjaei, and A. V. Papadopoulos, "Real-time containers : A survey," 2nd Workshop on Fog Computing and the IoT, vol. 80, p. 9, Jan. 2020, doi: 10.4230/oasics.fog-iot.2020.7.

[6] K. Sandström, A. Vulgarakis, M. Lindgren and T. Nolte, "Virtualization technologies in embedded real-time systems," 2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA), Cagliari, Italy, 2013, pp. 1-8, doi: 10.1109/ETFA.2013.6648012.

[7] H. P. Breivold and K. Sandström, "Virtualize for test environment in industrial automation," Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA), Barcelona, Spain, 2014, pp. 1-8, doi: 10.1109/ETFA.2014.7005089.

[8] G. Heiser, "The role of virtualization in embedded systems," Proceedings of the 1st Workshop on Isolation and Integration in Embedded Systems, pp. 11–16, Apr. 2008, doi: 10.1145/1435458.1435461.

[9] M. Wahler, R. Eidenbenz, C. Franke and Y. Pignolet, "Migrating legacy control software to multi-core hardware," in 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), Bremen, Germany, 2015 pp. 458-466. doi: 10.1109/ICSM.2015.7332497

[10] A. Moga, T. Sivanthi, and C. Franke, "OS-Level Virtualization for Industrial Automation Systems: Are We There Yet?," Proceedings of the 31st Annual ACM Symposium on Applied Computing, pp. 1838–1843, Apr. 2016, doi: 10.1145/2851613.2851737.

[11] T. Goldschmidt, S. Hauck-Stattelmann, S. Malakuti, and S. Grüner, "Container-based architecture for flexible industrial control applications," Journal of Systems Architecture, vol. 84, pp. 28–36, Mar. 2018, doi: 10.1016/j.sysarc.2018.03.002.

[12] T. Tasci, J. Melcher and A. Verl, "A Container-based Architecture for Real-Time Control Applications," 2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC), Stuttgart, Germany, 2018, pp. 1-9, doi: 10.1109/ICE.2018.8436369.

[13] M. Cinque, R. Della Corte, A. Eliso, and A. Pecchia, "RT-CASES: Container-Based Virtualization for Temporally Separated Mixed-Criticality Task Sets.," Euromicro Conference on Real-Time Systems, vol. 133, no. 133, Jul. 2019, doi: 10.4230/lipics.ecrts.2019.5.

[14] J. Wu and T. -I. Yang, "Dynamic CPU allocation for Docker containerized mixed-criticality real-time systems," 2018 IEEE International Conference on Applied System Invention (ICASI), Chiba, Japan, 2018, pp. 279-282, doi: 10.1109/ICASI.2018.8394587.

[15] V. Ibarra-Junquera, A. González, C. M. Paredes, D. Martínez-Castro and R. A. Nuñez-Vizcaino, "Component-Based Microservices for Flexible and Scalable Automation of Industrial Bioprocesses," in IEEE Access, vol. 9, pp. 58192-58207, 2021, doi: 10.1109/ACCESS.2021.3072040.

[16] B. Almadani, N. Alshammari and A. Al-Roubaiey, "Adaptive Cruise Control Based on Real-Time DDS Middleware," in IEEE Access, vol. 11, pp. 75407-75423, 2023, doi: 10.1109/ACCESS.2023.3296317.

[17] R. S. Auliya, C.-C. Chen, P. H. Lin, D. Liang, and W.-J. Wang, "Optimization of message delivery reliability and throughput in a DDS-based system with per-publisher sending rate adjustment," Telecommunication Systems, Aug. 2023, doi: 10.1007/s11235-023-01045-x.

[18] F. Reghenzani, G. Massari, and W. Fornaciari, "The Real-Time Linux kernel," ACM Computing Surveys, vol. 52, no. 1, pp. 1–36, Feb. 2019, doi: 10.1145/3297714.

[19] C. M. Paredes, D. Martínez-Castro, V. Ibarra-Junquera, and A. González, "Detection and Isolation of DoS and Integrity Cyber Attacks in Cyber-Physical Systems with a Neural Network-Based Architecture," Electronics, vol. 10, no. 18, p. 2238, Sep. 2021, doi: 10.3390/electronics10182238.

[20] G. Rezende Alles, A. Carissimi and L. Mello Schnorr, "Assessing the Computation and Communication Overhead of Linux Containers for HPC Applications," 2018 Symposium on High Performance Computing Systems (WSCAD), Sao Paulo, Brazil, 2018, pp. 116-123, doi: 10.1109/WSCAD.2018.00027.

[21] Á. Kovács, "Comparison of different Linux containers," 2017 40th International Conference on Telecommunications and Signal Processing (TSP), Barcelona, Spain, 2017, pp. 47-51, doi: 10.1109/TSP.2017.8075934.

[22] DDS Foundation, "What is DDS?," DDS Foundation, 2023. Available: https://www.dds-foundation.org/what-is-dds-3/. [Accessed: Sep., 2023]

[23] Industrial shields. Available: https://www.industrialshields.com/ [Accessed: Sep., 2023]

[24] ROS, "rosbridge_suite," *ROS Wiki*, Sep. 07, 2022. Available: http://wiki.ros.org/rosbridge_suite. [Accessed: Sep., 2023]