IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 26 July 2023, accepted 13 August 2023, date of publication 17 August 2023, date of current version 23 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3305951

RESEARCH ARTICLE

Finite-Time Consensus and Readjustment Three-Stage Filter for Predictive Schedules in FMS

ALEX LUIZ DE SOUSA[1] AND ANDRE SCHNEIDER DE OLIVEIRA[2], (Member, IEEE)
[1]Department of Information Systems, Santa Catarina State University (UDESC), São Bento do Sul, Santa Catarina 89283-081, Brazil
[2]Graduate Program in Electrical and Computer Engineering, Federal Technological University of Paraná (UTFPR), Curitiba, Paraná 80230-901, Brazil

Corresponding author: Alex Luiz de Sousa (alex.sousa@udesc.br)

ABSTRACT Enabling manufacturing systems to adapt quickly to production needs is crucial for industries to gain a competitive advantage. Modern manufacturing systems must be sufficiently flexible and intelligent to react promptly to market demand and ensure that manufacturing operations are deadlock-free. This study proposes a distributed multi-agent system (MAS) for flexible manufacturing systems (FMS) with order-controlled production that avoids deadlocks through a readjustment three-stage filter. Production schedules generated by a classical predictive scheduling algorithm are modified by three readjustment filters to avoid potential deadlocks and are used by MAS agents to drive production. The first filter sequentially groups certain events from schedules according to the buffer type of their respective production resources. The second filter parallelizes certain events associated with null-buffer resources and breaks the contiguity of other specific events to optimize production. The third filter identifies overlap by mapping the events that cause deadlocks and readjusting their positions in the production schedules. The readjusted schedules are sent to the MAS agents who cooperate in executing production tasks using event-based consensus control with predefined times and message exchanges via robot operating system topics. A physical factory simulation platform is modified to operate as an FMS and used to evaluate the proposed approaches. The FMS simulates a production environment with resource sharing and product parallelism. The experiments prove that the rescheduled predictive schedules are fully executed in the FMS without employing computationally onerous rescheduling methods for deadlock situations involving non-preemption and circular waiting. This study details the operations of consensus control and the algorithms used in the readjustment filters of the predictive schedules. At the end of the paper, information is provided on the MAS load scalability and the time complexity of the proposed agent model and the readjustment filters.

INDEX TERMS Multi-agents, flexible manufacturing system, predictive scheduling, readjustment, industrial automation.

I. INTRODUCTION

Owing to technological advances and market competitiveness, industries are creating increasingly complex manufacturing environments to meet customer demands for customization and responsiveness. In modern environments, industries view flexibility in the production process as a strategy to obtain competitive advantage. To ensure flexibility and reconfiguration, modern manufacturing systems must evolve their architecture in terms of intelligence [1]. Thus, many researchers have focused on developing intelligent architectures for flexible manufacturing systems (FMS), relying on concepts and technologies aligned with the context of Industry 4.0. FMSs are being updated and transformed by the internet of things (IoT), big data, cyber-physical systems, virtual reality, cloud computing, and various other Industry 4.0 technologies that have emerged to improve flexibility across the system [2].

An FMS can be characterized as a form of production with high levels of automation that is able to adapt and react promptly to production demands or in the face of unforeseen situations (e.g., rush orders and deadlocks). Flexibility in an FMS is observed through the ability of the system to change settings and adapt quickly to production requirements [3].

The associate editor coordinating the review of this manuscript and approving it for publication was Zhiwu Li.

However, as the variety of products processed in parallel and the machines and resources used in production processes increase, the control of FMSs becomes increasingly complex. The challenges of introducing adaptive and flexible manufacturing control approaches can be solved using distributed systems supported by cooperative autonomous units [4].

One distributed control strategy that can be exploited for coordinating complex systems is multi-agent control [5]. Multi-agent systems (MAS) are exceptionally well-suited for modeling complex problems because of the possibility of breaking the problem down into smaller parts and encapsulating the functionality. The MAS represents a promising approach for developing controls for FMS because of its decentralized and cooperative manner in which agents solve optimization problems and handle complex tasks [6]. Working cooperatively requires agents to communicate and reach a ''consensus'' [7], [8].

A consensus can be viewed as an algorithmic procedure that allows state convergence between locally autonomous agents collaborating toward a common goal [9]. According to Weiss [10],negotiation, argumentation, voting, auctions, and coalitions are the methods used to achieve consensus in cooperative and competitive scenarios. Consensus is the basis of the distributed coordinated control of an MAS [11]. However, according to Mezgebe et al. [9], the applicability of consensus for control problems in manufacturing, specifically for scheduling problems, has yet to be thoroughly studied. Consensus algorithms have rarely been adapted as decision-making algorithms or are yet to be implemented in the FMS [9]. However, with a suitable consensus method, the agents can cooperate in the execution of manufacturing tasks defined by scheduling.

Production scheduling is a classic topic in manufacturing control and is crucial for improving resource utilization and FMS efficiency [12], [13]. Many researchers have focused on developing new production scheduling methods. However, an efficient scheduling approach for a particular environment, which may involve many manufacturing resources and specific integrated interactions, can be difficult and sometimes impossible to reproduce in other scenarios. However, classical scheduling algorithms combined with other production control methods can facilitate the adaptation of this approach to other environments. Job shop scheduling (JS) is one of the most explored production scheduling problems. The JS is a combinatorial optimization problem that aims to assign a set of jobs to a set of machines, typically to minimize the maximum completion time for all jobs [14], [15]. Scheduling problems in FMS are best handled by a variant of JS known as flexible job shop scheduling (FJS). The FJS assumes that any machine can process an operation from a given set of available machines [16], [17], [18].

Although many scheduling problems have been extensively researched, some studies have considered infinite buffer spaces in manufacturing systems to reduce the complexity of solutions [19], [20]. However, in most real-world scenarios, unlimited buffering is not a realistic expectation because there may be physical limits to the resource capacity [17]. Consequently, issues such as resource overload and deadlocks are ignored, thereby incurring control problems and restricting the flexibility of the FMS. Additionally, environments capable of producing different products on the same machine are increasingly present in the industry, allowing parallel production flows. This implies that tasks from different production flows can compete for shared resources, leading to concurrent and potentially deadlock scenarios in the FMS. Therefore, it is critical to improve the proposed control solutions by considering buffers, product parallelism, and adaptations to different scenarios. According to Gao et al. [19], automated manufacturing systems, especially the FMS, can be regarded as a generalization of the FJS with limited capacity buffers.

A significant difficulty to controller design of FMSs is the complexity of these systems due to the stochastic behavior characterized by random processing times, product variability, the high number of simultaneous events, deadlocks, and production planning under uncertainty. Therefore, it is a crucial point for industries where specific scheduling problems can be solved directly in the generated predictive schedules, optimizing, prioritizing, and implementing actions to increase productivity without interfering with the control of FMSs. These issues are complex to be concerned with during production and can be better solved previously in production planning.

This paper presents an applied study of a physical factory simulation platform modified to operate as an FMS. Production on the platform is order controlled, and a distributed MAS executes production according to the schedules generated by a classical predictive scheduling algorithm. Agents cooperate using consensus control based on processing events at predefined times, and exchange messages between agents via robot operating system (ROS) topics. This study aims to ensure that different production schedules defined based on a complete view of the FMS state are executed deadlock-free by the MAS. Each schedule comprises a set of events representing tasks in the manufacturing resources of given products or parts. A novel method involving a readjustment three-stage filter was employed to produce schedules to ensure flexibility and reduce deadlocks in the system. The first method readjusts schedules by grouping certain events according to the buffer type. The second method parallelizes certain null buffer events and breaks the contiguity of other specific events to optimize the production schedules. Finally, the third method identifies overlaps by mapping the events that cause deadlocks and readjusting their positions in production schedules. The proposed methods are complementary and act as filters for schedules as defined by classical predictive scheduling. These methods ensure deadlock-free schedules for MAS operations and improve the responsiveness of FMS. According to Mezgebe et al. [9], the agents' skills can be used in scheduling to improve the

quality of the final solutions. The MAS developed for the FMS under study replaced the original platform's centralized fixed production cycle control.

The remainder of this paper is organized as follows: Section II presents the related studies. Section III presents the problem statements and assumptions. Section IV presents the proposed MAS architecture using consensus and agent models. Section V explains the factory simulation platform modified for the FMS concepts. Section VI presents the filters used for deadlock-free production. Section VII discusses the experiments. Section VIII presents an overall evaluation of the approach and provides information on the time complexity and scalability of the system. Finally, Section IX presents the conclusions and future work.

II. RELATED WORK

According to Hansmann and Hoeck [21], an FMS is a highly automated form of a multifunctional production facility, such as a job shop facility. The small-scale and custom manufacturing capabilities of job shops combined with the technologies and adaptive production flexibility of the FMS create a powerful manufacturing environment. With the support of a distributed MAS, an FMS can become more efficient owing to the segmentation of production tasks, greater tolerance to faults owing to the decentralization of control, and support functionalities (e.g., cognition capacity), among other advantages. Multiagents make distributed control more scalable, adaptable, flexible, and robust [22]. However, the flexibility and efficiency of FMS also depend on deadlock-free scheduling.

Fazlirad and Brennan [23] presented the primary multi-agent scheduling schemes suggested in the literature. According to researchers, agent-based scheduling approaches are best applied when the environment is complex, flexible, dynamic, and subject to frequent interruptions. Meilanitasari and Shin [24] also reviewed other modern prediction and optimization methods in job shop scheduling for the FMS. However, researchers have identified that the search for greater agility and flexibility in some newly developed FMSs results in many uncertainties that must be quantified and modeled explicitly. Uncertainties can cause unexpected events that can negatively affect the expected performance metrics. Uncertainties and interruptions are undesirable conditions in manufacturing systems, because they reduce productivity and cause industrial losses.

The achievement of deadlock-free production schedules for complex systems is a promising research topic. In an FMS, the probability of deadlock is high because of resource sharing, interoperability, and the need for reconfiguration. Therefore, the system architecture must be sufficiently intelligent to identify and avoid deadlocks. Examples of deadlocks are delays caused by failed scheduling processes and complete production stoppage (deadlock), which limit the production process and controllability of the system. Deadlock can also refer to the period during which a product remains in the processing machine, resulting in the inability of the machine to perform any other work [20], [25]. Bi et al. [26] reported that unexpected interruptions (e.g., resource failures) also occur in dynamic manufacturing systems, causing the production schedule to not be executed entirely.

According to the literature, several studies present meaningful solutions for certain deadlock situations. Research involving dynamic scheduling that proposes methods for recovery from deadlocks can be found in [14], [18], [27], [28], [29], [30], [31], [32], [33], and [34]. These approaches generally employ heuristic or artificial intelligence techniques such as evolutionary algorithms or neural networks to identify the most viable rescheduling plan. However, the number of methods used to reconstruct scheduling is limited. This is because of the lack of available information regarding the state of the system, which is only known during its execution. However, classical predictive scheduling contains complete, at least initially, information on the system conditions to generate production schedules.

Another approach widely applied in dynamic and flexible systems is dispatching rules (DR), regarded as ''scheduling decisions'' to solve production problems. Teymourifar et al. [35] proposed an approach for extracting efficient rules for a type of FJS where jobs arrive at different times in the system. Machine crashes that occurred stochastically under buffer-limited conditions were considered in this study. Researchers have combined a gene expression programming (GEP) method with a simulation model to design DRs into scheduling policies. Durasević and Jakobović [29] had similar objectives to Teymourifar's; however, the researchers proposed a method divided into two parts to create dynamic DRs. In Luo's [30] research, a method using DQNs trained with enhanced Deep Q-learning (DQL) was employed to determine the most suitable DR from a set of available DRs for scheduling problems. However, the inclusion of DRs that depend on information regarding the states of buffers, products, resources, and runtime complicates dynamic scheduling approaches. In addition, specific approaches require training or learning, which is computationally expensive.

Some approaches for solving deadlocks seek to swap tasks between machines (and buffers). In the literature, there are two versions of JS problems with a blocking constraint: blocking without swapping, and swapping. In the case of blocking without swapping, the buffer-capacity constraint (or infinite buffer) was replaced by a zero-capacity constraint. Research on blocking job shop (BJS) problems can be found in [17], [36], and [37]. Generally, methods for dealing with BJS involve metaheuristics, such as Simulated Annealing and Tabu Search, which are enhanced to explore feasible scheduling solutions. However, combining these methods with feasibility recovery strategies is necessary because of the large number of infeasible solutions obtained during the search. Mogali et al. [17] argued that the feasibility recovery step slows down the algorithm considerably, resulting in high

times for exploring small areas of the search space. Additionally, the resources required to perform job swapping must be considered to resolve deadlocks. However, similar to the rail systems studied by Sasso et al. [38], other manufacturing systems have constraints that prevent job exchanges, such as a lack of buffers or reduced mobility in transporting items between manufacturing resources.

Other researchers adopted Petri nets (PN) models to design FMSs and develop deadlock avoidance policies, as in [13], [25], [39], [40], [41], [42], [43], [44], and [45]. Several approaches are based on siphon detection (structural blockages), as in [39] and [43]. Kaid et al. [43] proposed a deadlock-control method based on strict minimal siphons (SMS). The control locations obtained using SMS were merged into a single control location using a colored PN. Bashir's [39] research proposed a method to calculate deadlock-free work zones and controls for a decentralized FMS supervisory structure. Evidently, avoiding deadlocks by applying constraints to mathematical models representing complex systems is challenging; however, the solution is restricted to specific models. Although there are efficient approaches to minimizing the structural complexity of supervisors modeled with PN, the requirements for preventing deadlocks may change or require new specifications. For example, when adding or removing resources, changing the capacity of a resource or adding a new product. According to Teymourifar et al. [35], many proposed solutions do not apply to real problems because of the assumptions made to simplify the models, which neglect essential features such as the carrying capacity or buffer space in the system.

Although there has been significant research on scheduling and deadlock control for complex manufacturing systems, several questions regarding the limitations and difficulties discussed in the literature should be considered. This study proposes a readjustment three-stage filter for predictive schedules to avoid deadlocks in flexible production environments with shared resources and product parallelism. The approach is relevant because it eliminates certain limitations and difficulties that other approaches are subject to, such as circular waiting and preemptive restrictions. The proposed method combines classical scheduling, pre-adjustment of predictive production schedules, and finite-time multi-agent consensus as an alternative to runtime rescheduling methods.

Finite-time multi-agent consensus is suitable for FMSs with order-controlled production because the production operations are sequential and their respective completion times are previously available in the predictive schedules. The compatibility between consensus and scheduling favors the predictability of the FMS and reduces the incidences of chance, increasing visibility over processes and greater control of production activities. Agreeing agents to a consistent FMS state at predefined time intervals by scheduling allows faster convergence and transient response, higher accuracy, and better rejection of disturbance or impasses. The methods of readjusting production schedules and task allocation for MAS can be regarded as a new scheduling form.

III. PROBLEM STATEMENT AND ASSUMPTIONS

This section discusses the important points for classifying this research problem. Many classical AI methods are centralized and cannot split a more extensively complex problem into smaller and simpler problems. In an FMS with distributed control, these methods are inefficient for operations with partial information and uncertainty. Therefore, rescheduling research typically focuses on designing new heuristic methods that process information in an FMS at runtime to recreate schedules when deadlocks occur. However, although some researchers have obtained efficient results, as in [12], [18], [20], [29], and [44], the proposed heuristics are usually limited to specific environments. Therefore, it is difficult to adapt them to other scenarios or compare them with other approaches.

Based on the purpose of the approaches reviewed in the literature, and in comparison with the proposal of the present study, two research fronts can be established that aim for deadlock-free production control in FMSs.

1) Research directed towards rescheduling with the generation of new production schedules at runtime.
2) Research on readjustment of predictive schedules to prevent potential impasses and deviations.

Table 1 presents specific characteristics for the two research fronts, briefly referred to as ''rescheduling'' and ''readjustment.'' The absence of approaches for the readjustment of predictive schedules for FMS in the literature leads to the assumption that approaches with this focus have not been explored or recognized with the importance they deserve. It is possible to characterize such research problems as semi-distributed (Table 1) problems by combining classical predictive scheduling approaches with distributed AI. Predictive scheduling algorithms are centralized and rely on complete system information. Despite obtaining production schedules with optimal makespan values, these schedules depend on fully meeting perfect environmental conditions. However, deadlocks, interruptions, and exceptions can cause deviations from the goals initially set by the FMS. In this case, distributed AI approaches (e.g., multiagents) can be employed to overcome scheduling limitations in production control.

TABLE 1. Properties of rescheduling and readjustment research.

	Rescheduling	Readjustment
Approach	Distributed	Semi-distributed
Information	Partial	Complete
Scheduling	Runtime	Predictive
Adaptation	Hard	Viable

Classic predictive scheduling algorithms are widespread and documented approaches, such as multiagent systems. Therefore, the adaptability of solutions using these approaches to other environments is feasible if there is a well-defined separation between scheduling, multi-agent

control, and intelligent approaches for the system. Otherwise, solutions remain restricted to specific environments and are difficult to compare or improve on other approaches in the literature. This research proposes a layered architecture, isolating scheduling, and production control from intelligent methods and functionality, which ensures that schedules are fully executed in the FMS.

IV. MAS ARCHITECTURE

The MAS architecture defined in this study is divided into layers, isolating the functionality and complexity of one layer from those of the others. To adapt the proposal to other production environments, the proposed scheduling and approaches were separated from the three-layer MAS architecture.

1) Communication layer: This layer uses the ROS framework to implement four topics for exchanging messages between agents. The topics ''synchrony,'' ''signaling,'' and ''notification'' are employed in the MAS consensus control that will be explained in subsection IV-B. Besides messages related to consensus, the topic ''notification'' is also used for messages between agents (e.g., to notify about the arrival of a part). The last topic is ''broadcast,'' used to receive events pertinent to production schedules stored in a queue-like data structure.
2) Control layer: Each agent is associated with one or more manufacturing resources, which can be manufacturing stations, machine tools, identification devices (NFC reader or color detection), or specific locations in the plant (e.g., part-delivery location). This layer is also responsible for executing the functionalities and decisions made by agents to control manufacturing resources. For example, moving or ejecting a product or starting an oven. This layer is responsible for accomplishing the tasks related to product manufacturing.
3) Physical layer: It provides hardware abstraction for the control layer to manage manufacturing resources through high-level commands executed by agents.

Fig. 1 presents an overview of the MAS architecture. Set $A = \{A_1, A_2, A_3, \ldots, A_i, \ldots, A_k\}$ to identify the agents in the control layer. The communication layer is identical for each agent A_1, except for some message types in the notification topic, specific to each agent, and the associated manufacturing resources. The physical layer at the bottom of the figure represents the manufacturing resources $(R_1, R2, R_3, \ldots, R_i, \ldots, R_n)$. Each agent is associated with at least one manufacturing resource and can manage it according to the production needs.

The communication layer provides services to the control layer, which uses the services from the physical layer. In addition to the ROS framework, other communication methods can be employed to exchange messages via topics (based on the publish-subscribe model) to the communication layer. For example, MQTT (Message Queuing Telemetry Transport) is a lightweight protocol over TCP/IP for exchanging messages between IoT devices that can operate with an MQTT broker; the broker is an intermediary entity that allows MQTT clients to communicate (similar to the ROS-based system).

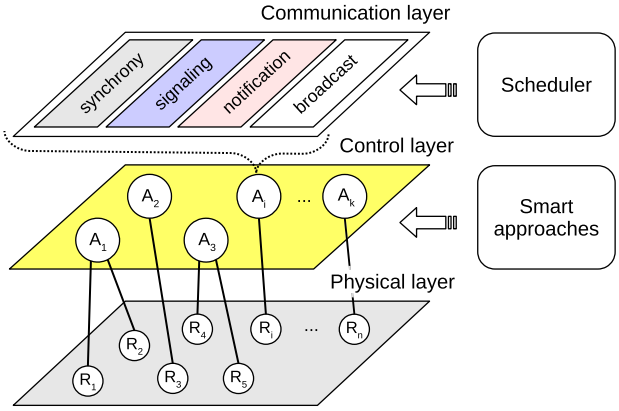

FIGURE 1. Three-layer MAS architecture overview.

The block referred to as ''smart approaches'' represents the individual or collective cognitive processes and functionalities of the agents (e.g., to reach a consensus), and the ''scheduler'' block defines the production schedules for the FMS.

A. SCHEDULER MODEL

Production scheduling is performed by the ROS node referred to as a ''scheduler,'' a software implementation that uses an FJS algorithm from the OR-Tools software package to generate the production schedules. A production schedule comprises f events, such that $E = \{E_0, E_1, \ldots, E_i, \ldots, E_f\}$. The scheduler sends production schedules to the MAS but does not participate in controlling the FMS as an agent.

An event is represented by five attributes: $E_i = [start, duration, agent, job, token]$.

- *Start*: represents the task start time.
- *Duration*: task duration time.
- *Agent*: ID of the agent that executes the task.
- *Job*: product ID.
- *Token*: sequence index for event control.

The start and duration attributes are given in seconds, beginning at 0 (zero) for the first production schedule event (event E_0). The agent attribute is unique and indicates the agent that processes the task. The job attribute identifies the product to which the event task is related so that a sequence of specific events defines the manufacturing of a product. The token attribute refers to a sequence index that indicates the events that can be processed at a given time, thus allowing the parallel processing of tasks by the agents indicated in the events that have the current token.

The following are the step-by-step processes performed by the scheduler to format and send production schedules to the MAS agents:

1) Load a properly formatted production model for the FJS algorithm.

2) Run the FJS algorithm for scheduling the model in production tasks.
3) Encapsulate the tasks into events with sequence indices to form the schedule.
4) Readjust the production schedule if deadlocks and deviations are identified.
5) Send the production schedule to the MAS agents.

First, the scheduler created or loaded a production model for the FJS algorithm. A production model comprises one or more lists of manufacturing resources, each containing a specific sequence of resources and their respective processing times. Each list is linked to a product via an ID so that the production model can contain multiple products. Next, the model was subjected to the FJS algorithm to perform scheduling. In this process, the algorithm fragments the manufacturing resource lists of the model into tasks, maintains the original sequence, and optimizes the completion time of all tasks.

Next, the scheduler encapsulates the tasks and other scheduling information into units defined as events. Each event in the schedule is assigned a sequence index that preserves task parallelism. Finally, the production schedules are sent to the MAS via the broadcast topic. Before sending, the schedule can still be adjusted to avoid deadlocks in the FMS (Section VI).

start →	0	5	0	0	0	
	5	5	1	0	1	← individual event
	10	5	7	0	2	
	10	5	0	1	2	
	15	5	1	1	3	
	20	5	8	1	4	← parallel events
	20	5	10	0	4	↙
			...			
end →	35	5	0	1	7	

FIGURE 2. Example of a production schedule.

Fig. 2 shows an example of a production schedule. Each row of the schedule corresponds to an event composed of the *start*, *duration*, *agent*, *job*, and *token* attributes. For an event with a *token* equal to 1, only the agent with ID 1 can perform the task. Thus, an event with a *token* equal to 3 must also be performed by an agent with ID 1. For events with tokens equal to 4, agents with IDs 8 and 10 can process the tasks defined for each *token* in parallel. The attribute data comprising scheduling events were used in the consensus mechanism developed for the MAS.

B. CONSENSUS MODEL

The consensus control of the present research can be classified as "predefined finite-time consensus," according to the consensus topologies for the MAS raised by [11] and [46]. In finite-time consensus, agents agree on a consistent state in a finite time interval. However, defining parameters correlating with the convergence time of MAS for the consensus protocol is challenging [46]. According to Li and Tan [11], the design and analysis of finite-time consensus algorithms are more complex than those of asymptotic consensus, whose convergence rate value is independent of time. However, for manufacturing systems, the convergence time of MAS should be more rigorous and be within a finite time to ensure predictability in production.

In finite-time predefined convergence, a specific case of finite-time consensus, the exact time to reach consensus is an a priori parameter in the consensus protocol [46]. The proposed consensus control is based on processed events in periods predefined by the production schedules generated for the studied FMS and the exchange of messages between the MAS agents. An event is triggered when agents update their control inputs upon receiving a synchronization signal. According to Dorri et al. [47], synchronization means that each agent's actions are aligned in time with those of the other agents. In the proposed consensus control, synchronization ensures that all the agents converge to a common token value, thereby characterizing a full-type consensus.

In the communication layer of the MAS architecture (see section IV), the topics "synchrony," "signaling," and "notification" are employed in the consensus. All MAS agents can read and write about these topics. Each agent only performs subscriber communication on the synchrony topic and updates the control inputs at these synchronization instants. Therefore, the frequency of communication and number of control updates are significantly reduced. Fig. 3 illustrates the operation of the proposed consensus control.

When the synchrony topic code block receives a synchronization message, the agents check whether the old token in the message is equal to the current token. If so, the agents update the current token with a new token value reported in the synchronization message. Otherwise, the token is already updated, and the agents under this condition are released from processing the current event until a new synchronization message is received. After the token is updated and a synchronization flag is reinitialized, the agents check whether their agent ID matches the ID contained in the current event. If so, the particular agent identified by the event processes the task by invoking production resources. The agent then removes the current event from the schedule and signals the completion of the task. However, the other agents also check if there is any control procedure for the same event (e.g., preparing a resource to receive the product). Finally, before removing the event from the schedule, the agents enable a blocking flag that prevents the token from being updated until the control procedure is complete.

When a task completion message is received in the signaling topic code block, the agents check whether it contains their agent ID. Otherwise, the agent waits for the lock flag to deactivate (WaitUnlock). Subsequently, the agent sends a status message that increments the value of the synchronization flag. This synchronization flag indicates the number of active agents that have checked the last event. However, when the task completion message contains the agent's ID, it also communicates with the increment the synchronization flag and waits for synchronism from the other agents (WaitSync).

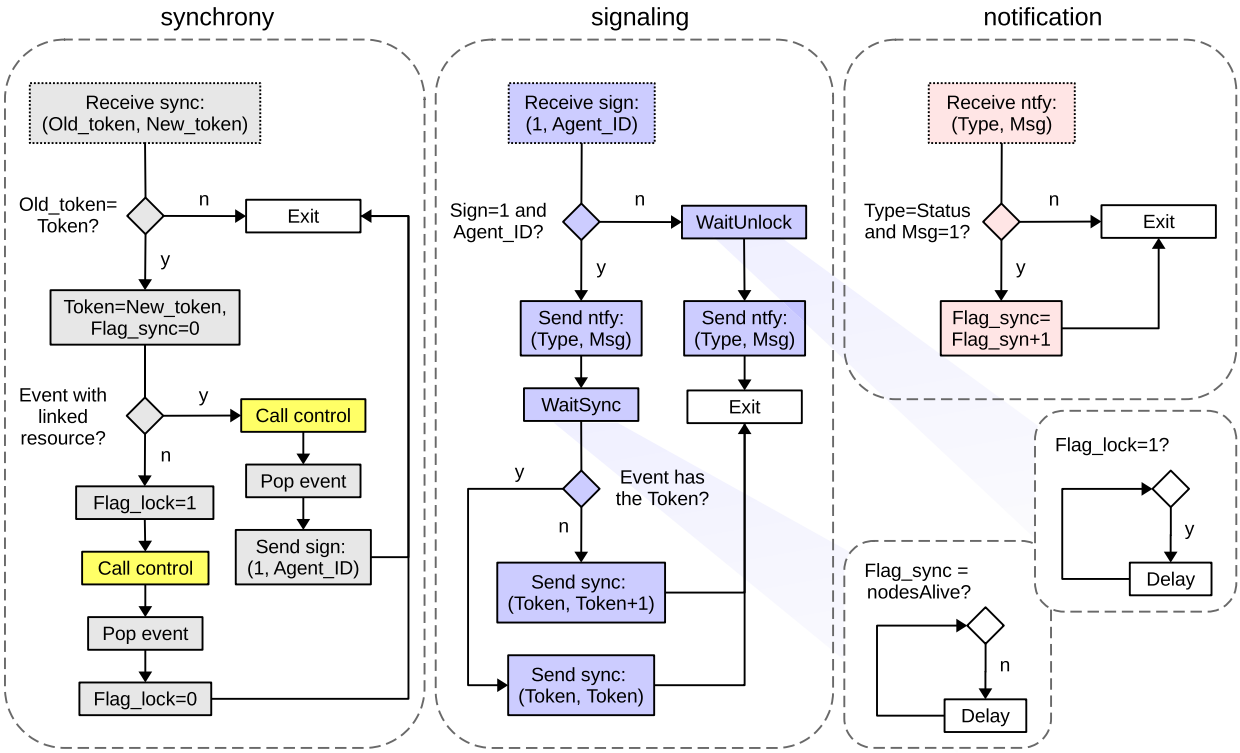

FIGURE 3. Proposed consensus control model.

If the current event does not have a token, the agent sends a message to the synchrony block informing the current and new tokens (token, token+1). Otherwise, if an event has the current token, the agent sends a synchronization message by repeating it (token or token). Notably, the current event is no longer the same as that used by the synchrony block because it has already been deleted. Therefore, if the current event has the same token, it is considered a parallel event to the production tasks defined for another MAS agent. This mechanism is necessary because the scheduler replicates the complete schedule for all the MAS agents. However, events that do not have an agent ID may still require the attention of specific agents (e.g., to notify that the product has been received).

Finally, in the notification block, when an agent receives a notification message of type ''status'' containing the value 1 (meaning ''on''), it increments its sync flag. Other messages are used in the notification block for agent communication but are specific to other procedures and have no relation to consensus. Whenever a new production schedule is started or restarted after its completion, the token is reset and the process of consensus and task execution is repeated. An essential aspect of this architecture is that the agent model performs signaling, synchronization, notification, broadcasting, and control in independent threads. Therefore, sequential processing does not delay the execution of schedules in individual or parallel events unless deadlocks or deviations are caused by manufacturing resources in the physical layer of the architecture.

C. AGENT MODEL

A well-structured agent model aligned with the consensus model guarantees scalability and synchronization with the FMS. Scalability is an important feature that should be considered at the design stage, because it is difficult to add it later to the system. Small and medium-sized enterprises often choose to invest in a category of FMS referred to as flexible manufacturing cells (FMCs), which have a lower financial cost, implement the cell concept, and can be interconnected with other FMCs or a larger FMS. Therefore, the MAS architecture must be sufficiently scalable to adapt an FMS to changes. With the complexity of FMSs, achieving a system balance with conventional control schemes is difficult.

Fig. 4 presents the proposed agent model, which consists of five threads running concurrently according to the support provided by the operating system embedded in the hardware. The synchrony, signaling, and notification threads are related to the topics and code blocks used for consensus, as discussed in Subsection IV-B. A broadcast refers to the messaging topic and the block responsible for receiving events for production schedules. Finally, the controller thread is related to the actions that an A_i agent performs on the resources associated with it, as well as individual or collective functionalities and decision making. The agent model favors the

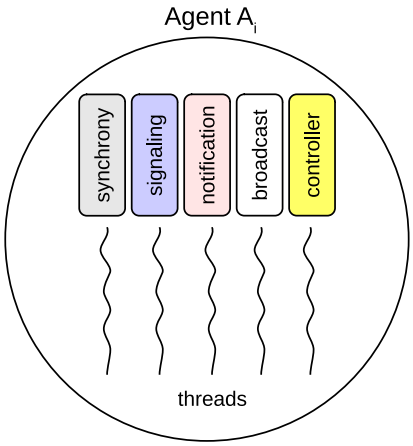

FIGURE 4. Proposed agent model.

system scalability because, in conjunction with the consensus model, it allows the inclusion, exclusion, or change of agents (and resources), independent of scheduling and production control.

Another essential characteristic of complex manufacturing systems is their ability to synchronize, because such systems usually comprise several interrelated subsystems. Similarly, FMS rely on synchronous points to track production tasks using different resource types. Moreover, production scheduling cannot be performed randomly, as production resources become available. There is no way to guarantee accurate responsiveness to demand regarding the production time, which makes adopting schedules with makespan predictions unfeasible. However, the agent and consensus models proposed for the MAS architecture of the FMS in this study guaranteed synchronism.

V. FACTORY SIMULATION PLATFORM

This section presents a physical factory simulation platform modified to operate as an FMS with order-controlled manufacturing[1]. The features of the FMS are described below.

- Vacuum gripper robot (VGR): A station with a robot and vacuum suction gripper for moving parts in the FMS.
- High-bay warehouse (HBW): An automated high-bay warehouse with 3 × 3 slots and containers for storing and retrieving parts.
- Multi-processing station (MPO): simulates an industrial oven and machining bench with a rotary table and a robot to move the parts.
- Sorting line and detection (SLD): The sorting line has a conveyor, part color detection chamber, and three pneumatic pistons.
- General-purpose manufacturing stations (MS1-MS7) consist of seven manufacturing stations with bays that accommodate parts and LEDs to indicate machining.

[1]Order-controlled manufacturing means production to meet customer demands or to stock, and may or may not be repetitive.

The factory also has a color-detection sensor (DCS), reader/writer near-field communication (NFC), parts disposal point (TSH), parts arrival point (DSI), and parts collection point (DSO). In total, 14 parts processing points (DSI, DSC, NFC, HBW, MPO, SLD, MS1, MS2, MS3, MS4, MS5, MS6, MS7, DSO) are employed in production planning. Fig. 5 shows the physical locations of the resources in the FMS.

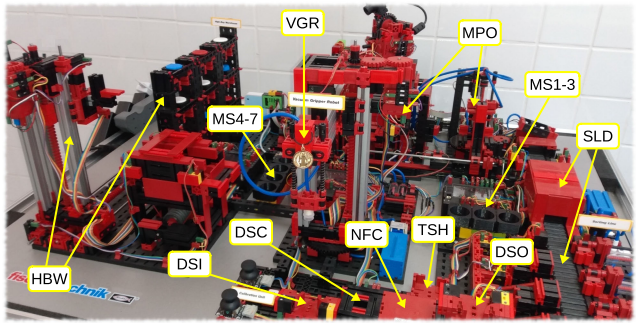

FIGURE 5. Simulation platform for the FMS.

Small cylindrical parts (height = 13.9 mm, diameter = 26.1 mm) of varying colors (white, red, and blue) were moved on the FMS to simulate production. Each part has an individual NFC identification tag that records the color and production history, along with date and time information.

The physical and functional characteristics of the manufacturing resources available on the platform allow them to be classified by the buffer type as follows:

- Limited buffer: holds a part indefinitely according to the capacity of the resource (e.g., containers in the HBW warehouse).
- Limited transient buffer: maintains part of the resource while the system processes other tasks (e.g., a manufacturing station).
- Null buffer: A non-buffered manufacturing resource that occupies the transport system during task processing (e.g., color detection sensor).

According to this classification, the DSO, TSH, and HBW resources are of the limited buffer type with a capacity of 1 (one part). The MS1, MS2, MS3, MS4, MS5, MS6, and MS7 resources are of the limited transient buffer type with a capacity equal to 1 because they can hold a newly machined part for a certain amount of time as long as the resource remains idle during the next token increment. Finally, the DSI, DSC, MPO, SLD, and NFC resources are of the null buffer type because they rely on the VGR robotic arm to suspend the part over the resource for identification. Thus, the robotic arm is unavailable for other parallel tasks until the DSI, DSC, MPO, SLD, or NFC release it.

In the FMS, manufacturing resources are associated with the agents of the proposed MAS architecture, which control the resources to accomplish the tasks defined in the production schedules. The MAS has 11 agents, identified from A_1 to A_{11}. Agent A_1 is responsible for transportation and coordinates the VGR robot to move parts to the FMS resources.

The DSI, DSC, NFC, DSO, and TSH are also associated with A_1 because the VGR controller is physically connected to them in the factory. Agent A_2 coordinates the MPO station resources, including a pneumatic actuator that pushes the part onto the SLD conveyor, which the VGR robot cannot reach. Agent A_3 coordinates the HBW robot to move parts from the rack to the conveyor belt and vice versa. Agent A_4 coordinates the SLD station resources to sort parts by color. Moreover, stations $MS1$-$MS7$ are coordinated by agents A_5-A_{11}. These stations have been added to the FMS to increase their potential as simulation platforms.

In practice, the VGR, MPO, HBW, and SLD station agents operate on the controllers associated with these stations. Stations $MS1$-$MS7$ had no controllers, and the agents ran independent processes on a computer that controlled the stations using a NodeMCU card. A Wi-Fi router or computer configured as a hotspot provides the FMS network. The scheduler and agents communicate via ROS topics. Production is coordinated according to the schedules defined by the scheduler node. The ROS framework was chosen for the topology because it is lightweight, general purpose, open source, and compatible with various industrial tools and products.

VI. READJUSTMENT THREE-STAGE FILTER

This section presents three methods for ensuring flexibility and reducing deadlock problems in the FMS under study. These methods are based on readjusting the predictive schedules, as discussed in Section III. The objective is to ensure that the production schedules generated by the FJS algorithm are executed entirely. The readjustment-focused approach is an alternative to computationally expensive rescheduling solutions, which are generally limited to specific environments.

A. SERIALIZER FILTER

The first method for deadlock-free production in the studied FMS is referred to as the "serializer filter." It identifies whether production schedules contain events associated with null-buffer resources. If so, the method serializes these events so that they do not occur in parallel with other events in the schedule. Otherwise, the VGR robot responsible for transporting parts in the FMS cannot move new parts to fulfill other scheduled tasks because it is busy with the current part.

Fig. 6 illustrates the operation of the serializer filter for a production schedule with two types of parts. The first part was manufactured according to the sequence $REF \prec DSI \prec DSC \prec NFC \prec MS1 \prec MS4 \prec DSO \prec REF$, whereas the second part was manufactured according to the sequence $REF \prec DSI \prec DSC \prec NFC \prec MS2 \prec MS5 \prec DSO \prec REF$. The numbered arrows indicate the movement of the loaded VGR robot (with the parts), whereas the dashed arrows represent the movement of the unloaded robot (without the parts). Given an initial production schedule (Fig. 6-A), the VGR robot must move from the reference point (REF) to the DSI to select the first part type (transition 0). The movement from REF to DSI corresponds to the first event in the schedule. Subsequently, the robot must return empty to REF (transition "a"), picking up the second type of part at the ISD (transition 1). However, the robot loaded with the first part taken from the ISD cannot be unloaded to pick up the second part, which causes a blockage in the FMS. Similarly, the problem is repeated in the DSC and NFC because of the null buffering of resources and sharing of transport robots. Predictive scheduling algorithms are generally not designed to identify or prevent this problem.

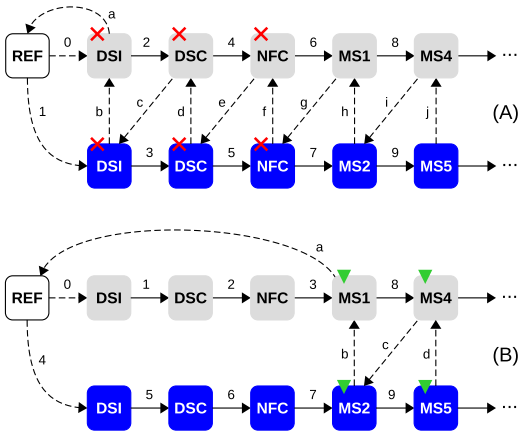

FIGURE 6. Operation of the serializer filter.

The serializer filter ensures that a part is processed sequentially on null buffer resources according to its manufacturing sequence until it is deposited on a buffer-limited or buffer-limited transient resource (Fig. 6-B). For the first part type, the VGR robot sequentially performed transitions 0 to 3 until it deposited the part at station MS1. MS1 and the other stations were classified as having a limited transient buffer (see Section V). Next, the VGR robot should move unloaded to REF (transition "a"), pick up a part of the second type (transition 4) and move it sequentially through the null buffer resources until it unloads the part at station MS2. Soon after, the VGR robot retrieves the part that was at MS1 (transition "b") and deposits it at MS4 (transition 8). Then it returns unloaded to pick up the other type of part left at MS2 (transition "c") and deposit it at MS5 (transition 9), continuing until the schedule ends. The serializer filter pseudocode is presented in Alg. 1.

Given a production schedule defined by list E of events and list B of agents associated with null-buffer resources, the serializer filter performs three functions to readjust the schedule: First, the serializer uses the *check_method*() function to check for null buffer resources in the schedule. It function checks the scheduled events for each part or product j until it finds an event in which the agent ID ($E_{i[agent]}$) is contained in B (lines 41×44). If found, the function includes an event in the *gpo* group (line 45) and searches for other events of the same type related to j. The search continues until the function finds an event j that does not have an agent ID associated with B, meaning that an event with limited buffer

Algorithm 1 - Serializer filter

```
1:  E ← [E_0, ..., E_i, ..., E_f]; B ← [0, 1, 2, 3, 4, 5, 6]
2:  function update_schedule()
3:      E_0[start], E_0[token] ← 0
4:      ctl ← [ ]
5:      for (i ← 0, |E|) do
6:          if (E_i[job] = E_i+1[job]) then
7:              E_i+1[start] ← E_i[start] + E_i[duration]
8:              E_i+1[token] ← E_i[token] + 1
9:              ctl ← [ ]
10:         else if (E_i[job] ! = E_i+1[job]) then
11:             if (E_i+1[job] ∈ ctl) then
12:                 E_i+1[start] ← E_i[start] + E_i[duration]
13:                 E_i+1[token] ← E_i[token] + 1
14:                 ctl ← [ ]
15:             else
16:                 E_i+1[start] ← E_i[start]
17:                 E_i+1[token] ← E_i[token]
18:                 ctl ← append(E_i[job])
19:                 ctl ← append(E_i+1[job])
20:             end if
21:         end if
22:     end for
23:     return E
24: end function
25: function exec_serializer(L)
26:     for (j ← 0, |J|) do
27:         for (g ← 0, |L_j|) do
28:             idx ← E.index(L_jg0)
29:             for (e ← 0, |L_jg|) do
30:                 E.remove(L_jge)
31:             end for
32:             for (e ← |L_jg|, 0) do
33:                 E.insert(idx, L_jge)
34:             end for
35:         end for
36:     end for
37:     return update_schedule()
38: end function
39: function check_method()
40:     L, aux ← [ ]
41:     for (j ← 0, |J|) do
42:         gpo ← [ ]
43:         for (i ← 0, |E|) do
44:             if (E_i[job] = j) and (E_i[agent] ∈ B) then
45:                 gpo ← append(E_i)
46:                 for (k ← i + 1, |E|) do
47:                     if (E_k[job] = j) and (E_k[agent] ∋ B) then
48:                         if (|gpo| = 1) and (gpo0_[agent] ∈ B) then
49:                             idx ← E.index(gpo0)
50:                             for (m ← idx − 1, 0) do
51:                                 if (gpo0_[job] = E_m[job]) then
52:                                     gpo.insert(0, E_m)
53:                                     break
54:                                 end if
55:                             end for
56:                         end if
57:                         aux ← append(gpo)
58:                         gpo ← [ ]
59:                     end if
60:                     break
61:                 end for
62:             end if
63:         end for
64:         if (|gpo| > 0) then
65:             aux ← append(gpo)
66:         end if
67:         L ← append(aux)
68:         aux ← [ ]
69:     end for
70:     if (|L| > 0) then
71:         return exec_serializer(L)
72:     else
73:         return E
74:     end if
75: end function
```

resources has been identified (lines 46 and 47). Next, the function checks whether the null buffer event included in the *gpo* was unique; there were no others to complete the group (at least one pair). In this case, the *gpo* event is sequentially linked to a previous event of the same *job*, such that in the execution of the method, the null buffer event is not isolated in the schedule. Thus, the previous event (E_m) is positioned at the beginning of the group, followed by the null buffer event (lines 48 to 56). Thus, if null buffer resources exist in the schedule under analysis, they are separated into two or more groups. The function assembles a list L with the respective events that must be processed sequentially for each product (lines 57 and 67). Therefore, schedule readjustment is necessary if this list contains null-buffer events (lines 70 and 71).

Function *exec_serializer*() adjusts the production schedule. For each part or product j, the function obtains the position index (idx) of the first event (L_{jg0}) of each group saved in L (lines 26×28). The function then removes all events of the group related to j from the initial schedule (lines 29 and 30) and inserts them again in reverse order, from last to first, from the position indices previously obtained (lines 32 and 33). Finally, the method calls the *update_schedule*() function to arrange the start ($E_{i_{[start]}}$) and tokens ($E_{i_{[token]}}$) for each event in the modified schedule (line 37). For this, the function compares whether the current event (E_i) and the next event (E_{i+1}) refer to the same product (line 6). If this is the case, the beginning of event E_{i+1} is updated to start after the end of event E_i (line 7). Similarly, the token of E_{i+1} is updated by 1 from the token value of E_i (line 8). However, if the current event E_i and the next event E_{i+1} refer to different products (line 10), the function checks whether the product (*job*) of the next event is contained in *ctl* (line 11). If it is, $E_{i+1_{[start]}}$ and $E_{i+1_{[token]}}$ (lines 12 and 13) are updated incrementally, and *ctl* is reset. Otherwise, updating the *start* and *token* for E_{i+1} involves repeating the respective values of E_i (lines 16 and 17). Finally, event-related product IDs are included in *ctl* for a new round of schedule updates (lines 18 and 19).

B. PRUNING FILTER

The second method for deadlock-free production in the studied FMS is referred to as the "pruning filter." This method seeks to identify null buffer events organized contiguously with limited or transient buffer events of the same product ($E_{i_{[job]}}$). If so, this method breaks the contiguity of the events. It readjusts the schedule such that the null buffer event is prioritized and arranged in parallel with events from other products. In other words, the method "prunes" the last limited buffer event contiguous to the same product's null buffer event. The pruned event obtains another token number and is positioned away from the null buffer event in the schedule. Although the FMS relies on the VGR robot to move the parts, the pruning filter can optimize some production schedules.

Fig. 7 illustrates the operation of the pruning filter for a production schedule with two types of parts. The first part

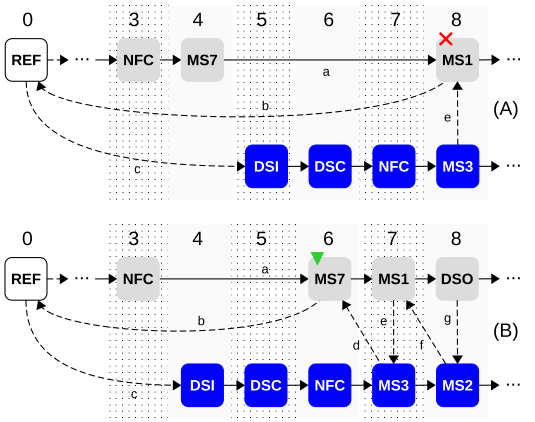

FIGURE 7. Operation of the pruning filter.

was manufactured according to the sequence $REF \prec DSI \prec DSC \prec NFC \prec MS7 \prec MS1 \prec DSO \prec REF$ and the second part was manufactured according to $REF \prec DSI \prec DSC \prec NFC \prec MS3 \prec MS2 \prec DSO \prec REF$. When running the production schedule without readjustment (Fig. 7-A), the VGR robot must leave the reference point (REF), pick up a part, and move it until it reaches the NFC. With the token was incremented to three, the VGR robot moved the NFC part to MS7. Next, the token was incremented to four, enabling the processing of the part at MS7. The VGR robot then unloads the part at station MS1 (transition "a"), which is the following resource in the current manufacturing sequence. The unloaded VGR robot goes to REF (transition "b") and picks up another part type at DSI (transition "c"). The token is incremented to 5, then to 6 and 7 until the part is deposited at MS3 (token 7). In summary, MS7 could process in parallel, but is in an event with an individualized token (Table 4). In this case, the schedule does not block but diverts the MAS control from the optimal sequence of part production, causing delays.

The pruning filter method avoids the deviation problem by parallelizing and prioritizing the task execution of events related to null buffer resources and separating contiguous buffer-limited events. Fig. 7-B explains the operation of the pruning filter method. For the first part type, the VGR robot sequentially executes events with tokens from 0 to 3 until it deposits the part at the MS7 station. The unloaded VGR robot picks up another part type at DSI (transitions "b" and "c") and moves the part to DSC (token 4) and NFC (token 5). With a token increment of 6, the NFC takes priority (null buffer resource) and processes until the part is deposited into MS3. The unloaded VGR robot is positioned at the MS7 station (transition "d"), waits for the part to be processed (token 6), and moves it to the next station (MS1). The remaining events followed the standard production control process performed by the MAS according to the schedule.

The pseudocode for the pruning filter is presented in Alg. 2. Being E, a list of events in a production schedule, and D, a list

Algorithm 2 - Pruning Filter

$E \leftarrow [E_0, \ldots, E_i, \ldots, E_f]$
$D \leftarrow [4, 7, 8, 9, 10, 11, 12, 13, 14, 15]$
function update_schedule()
    $E_{0_{[start]}}, E_{0_{[token]}} \leftarrow 0$
    $ctl \leftarrow [\,]$
    for $(i \leftarrow 0, |E|)$ **do**
        if $(E_{i_{[job]}} = E_{i+1_{[job]}})$ **then**
            $E_{i+1_{[start]}} \leftarrow E_{i_{[start]}} + E_{i_{[duration]}}$
            $E_{i+1_{[token]}} \leftarrow E_{i_{[token]}} + 1$
            $ctl \leftarrow [\,]$
        else if $(E_{i_{[job]}} != E_{i+1_{[job]}})$ **then**
            if $(E_{i+1_{[job]}} \in ctl)$ **then**
                $E_{i+1_{[start]}} \leftarrow E_{i_{[start]}} + E_{i_{[duration]}}$
                $E_{i+1_{[token]}} \leftarrow E_{i_{[token]}} + 1$
                $ctl \leftarrow [\,]$
            else
                $E_{i+1_{[start]}} \leftarrow E_{i_{[start]}}$
                $E_{i+1_{[token]}} \leftarrow E_{i_{[token]}}$
                $ctl \leftarrow append(E_{i_{[job]}})$
                $ctl \leftarrow append(E_{i+1_{[job]}})$
            end if
        end if
    end for
    return E
end function
function exec_pruning(L)
    for $(d \leftarrow 0, |L|)$ **do**
        $idx \leftarrow E.index(L_d)$
        for $(k \leftarrow idx + 1, |E|)$ **do**
            if $(L_{d_{[job]}} = E_{k_{[job]}})$ **then**
                $E.insert(k, L_d)$
                $E.remove(L_d)$
                break
            end if
        end for
    end for
    return $update\_schedule(L)$
end function
function check_method()
    $L \leftarrow [\,]$
    for $(i \leftarrow 0, |E|)$ **do**
        if $(E_{i_{[agent]}} \in D)$ and $(E_{i-1_{[agent]}} \ni D)$ **then**
            if $(E_{i_{[job]}} = E_{i-1_{[job]}})$ and $(i > 0)$ **then**
                $L \leftarrow append(E_i)$
            end if
        end if
    end for
    if $(|L| > 0)$ **then**
        return $exec\_pruning(L)$
    else
        return E
    end if
end function

of agents associated with resources of buffer limited or buffer transient limited, the pruning filter performs three functions for schedule readjustment. First, the *check_method*() function (line 39) goes through the scheduled events to identify whether the event agent belongs to D and, if so, whether the event preceding it belongs to D (line 42). When this condition is satisfied, the function determines whether the product or part ($E_{i_{[job]}}$) identified in the task of the current event is the same as that of the previous event ($E_{i-1_{[job]}}$) (line 43). If so, the current event is reserved in list L (line 44), and the process is repeated until all the events in the schedule have been checked. Finally, if the list L contains events, the production schedule must be readjusted (lines 48 and 49).

The function *exec_pruning*() readjusts the production schedule (line 26). For each event loaded at L, the function obtains its respective index (*idx*) from its position in the initial schedule (line 28). The function then proceeds through the rest of the schedule from each index obtained to identify the next event with the same product ($E_{i_{[job]}}$) as the L_d event under analysis (Lines 29 and 30). If found, event L_d is inserted into the schedule at the position defined by k (line 31) and the old event is deleted (line 32). Finally, the method calls the *update_schedule*() function to arrange the starts ($E_{i_{[start]}}$) and tokens ($E_{i_{[token]}}$) for each event in the modified schedule (line 37). This update function operates in the same manner as the serializer filter method.

C. OVERLAY FILTER

Previous filters can avoid deadlocks and optimize some predictive schedules but cannot prevent overlaps in manufacturing resources. In this study, overlap is understood as an attempt to process the part of a resource already occupied by another. This type of deadlock occurs because the resources are buffer-limited, and the system must be able to avoid them to ensure flexibility. The third filter, called the "overlay filter," can identify and prevent overlaps in production resources. This method moves certain overlapping events into the schedule to avoid deadlocks.

Fig. 8 illustrates the operation of the overlay filter for a production schedule with two types of parts. The first type was manufactured according to the sequence $REF \prec DSI \prec MS7 \prec MS5 \prec MS1 \prec DSC \prec DSO \prec REF$, and the second type was based on $REF \prec DSI \prec MS7 \prec DSC \prec MS1 \prec MS3 \prec DSO \prec REF$. First, the original schedule was subjected to serialization and pruning filters, resulting in the schedule shown in Fig. 8-A. However, the readjusted schedule contains deadlocks because of the overlaps highlighted in $DSI \rightarrow MS7$ (token 2) and $DSC \rightarrow MS1$ (token 4) for the second type of schedule. In $MS7$, the overlap occurs because the VGR robot would deposit a part of the first type in the resource (token 1) and then fetch a part of the second type in DSI (transition "a" token 1) also to attempt to deposit in MS7 (token 2). However, $MS7$ would already be busy with the first part type, and the system would enter a deadlock state owing to the circular waiting. Similarly, the overlap would also occur at $MS1$, which would be busy with a part of the first type (transition "c," token 3) when the VGR robot would go to deposit the part of the second type at the same resource (token 4). The overlay filter identifies overlapping events while generating a mapping of state transitions. Table 2 lists the indices of the overlapping events and the mapping performed for the schedule shown in Fig. 8-A.

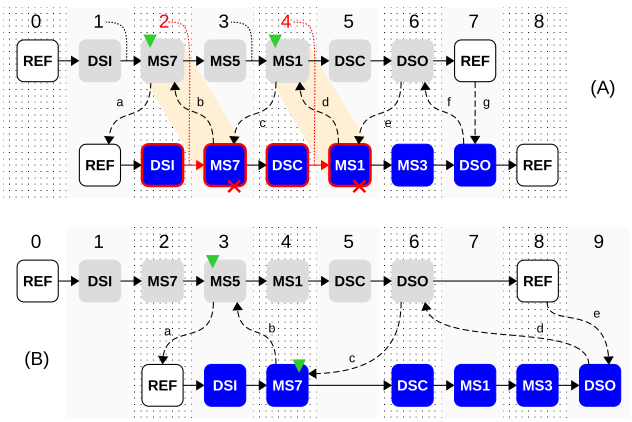

FIGURE 8. Operation of the overlay filter.

TABLE 2. Mapping of state transitions and overlap indexes.

```
»»» generate overlay mapping:
REF -> DSI [0]   [0, 5, 0, 0, 0] -> [5, 5, 1, 0, 1]
DSI -> MS7 [0]* [5, 5, 1, 0, 1] -> [10, 5, 13, 0, 2]
MS7 -> REF [a]
REF -> DSI [1]   [5, 5, 0, 1, 1] -> [10, 5, 1, 1, 2]
DSI -> MS7 [1]S [10, 5, 1, 1, 2] -> [15, 5, 13, 1, 3]
MS7 -> MS5 [0]   [10, 5, 13, 0, 2] -> [15, 5, 11, 0, 3]
MS5 -> MS1 [0]* [15, 5, 11, 0, 3] -> [20, 5, 7, 0, 4]
MS1 -> MS7 [b]
MS7 -> DSC [1]   [15, 5, 13, 1, 3] -> [20, 5, 2, 1, 4]
DSC -> MS1 [1]S [20, 5, 2, 1, 4] -> [25, 5, 7, 1, 5]
MS1 -> DSC [0]   [20, 5, 7, 0, 4] -> [25, 5, 2, 0, 5]
DSC -> DSO [0]   [25, 5, 2, 0, 5] -> [30, 5, 14, 0, 6]
DSO -> MS1 [c]
MS1 -> MS3 [1]   [25, 5, 7, 1, 5] -> [30, 5, 9, 1, 6]
MS3 -> DSO [1]   [30, 5, 9, 1, 6] -> [35, 5, 14, 1, 7]
DSO -> DSO [d]
DSO -> REF [0]   [30, 5, 14, 0, 6] -> [35, 5, 0, 0, 7]
DSO -> REF [1]   [35, 5, 14, 1, 7] -> [40, 5, 0, 1, 8]
   overlap idx: [6, 10]
```

The generated mapping shows the overlay transitions (identified by "S"), the transitions for events with resources that will be busy with parts (specified by "*"), the part IDs (in brackets), and the schedule overlay indexes (identified in *idx*). The strategy of the overlap filter is to locate the index (called the *wex*) of the event with the same agent ID that precedes the overlap event and swap its position to avoid a deadlock. The *wex* index event is inserted immediately after the event with the same *job* that precedes the *wex* in

the schedule (an index called *fdx*). This strategy of swapping positions preserves the originally defined manufacturing sequence of the parts. Fig. 8-B shows the readjusted- and overlay-free schedule. The overlap filter readjusts the schedule so that the transition "a" for the first part type does not occur in $MS7$, which is shared with the second part type. Consequently, when the VGR robot moved to the second type, $MS7$ was free. However, the overlap that would occur at $MS1$ was also resolved by switching the event position (token 7) for the second part type.

The pseudocode for the overlay filter is presented in Alg. 3. R is a list of the names of the fabrication resources used for mapping, B is a list of agents associated with buffer-limited resources, and A is a list of letters used to identify transitions (lines 1 to 3). M, L, and K are variables for saving, mapping data, overlay index data, and resource names used in the decision control (line 4). Initially, the overlay filter uses the function *check_method*() (line 31) to check for an overlap. Thus, the indices of the overlapping events are stored in L (line 55). The intermediate pseudocode (lines 34×52) refers to the mapping of the transitions for the limited buffer resources occupied by parts (line 42), normal processing transitions (line 45), and overlay transitions (line 54). Then, the mapping is available (lines 61 and 62), along with the possible overlap indices stored in L (line 64). If L is not null, then one or more overlaps are identified during mapping.

The *fix_overlay*() function readjusts the production schedules that contain overlaps (line 5). First, the function takes the index of the event of the same agent ID as the overlapping event and saves it to *wex* (lines 7-12 if *wex* precedes L_i in the schedule, or lines 13-18 if *wex* succeeds L_i). Next, the function obtains the index of the event preceding *wex* and saves it in *fdx* (lines 19×24). The *wex* index event is shifted and inserted immediately after the *fdx* index event (lines 25×27). This code sequence is executed for each overlay index contained in L. Finally, the schedule is updated with the *update_method*() function, which processes E in the same manner as the serializer and pruning filter functions. An essential aspect of the overlay filter was omitted in Alg. 3 to simplify the presentation that its functions are recursive. Therefore, the schedule was readjusted and checked successively until it was free of overlap. Moreover, in each round of schedule readjustments, the method applies serializer and pruning filters to avoid deadlocks due to the displacement of the *wex* index events.

The schedule can be sent to the MAS at the end of all readjustments. Using these methods as filters enables the addition of new filters to the solution as required without affecting the MAS architecture. The serialization, pruning, and overlay filters proposed in this study reduce the possibility of deadlocks in the FMS, and prevent deviations that affect the responsiveness of the system.

Algorithm 3 - Overlay filter

```
1:  R ← [REF, DSI, DSC, ..., MS6, MS7, DSO]
2:  B ← [7, 8, 9, 10, 11, 12, 13]
3:  A ← [ASCII_letters]; a ← 0
4:  M, L, K ← [ ]
5:  function fix_overlay()
6:      for (i ← 0, |L|) do
7:          for (j ← L_i − 1, 0) do
8:              if (E_{L_i[agent]} = E_{j[agent]}) then
9:                  wex ← E.index(E_j)
10:                 break
11:             end if
12:         end for
13:         for (j ← L_i + 1, |E|) do
14:             if (E_{L_i[agent]} = E_{j[agent]}) then
15:                 wex ← E.index(E_j)
16:                 break
17:             end if
18:         end for
19:         for (e ← wex − 1, 0) do
20:             if (E_{wex[agent]} = E_{e[agent]}) then
21:                 fdx ← E.index(E_e)
22:                 break
23:             end if
24:         end for
25:         aux ← E_wex
26:         E.remove(E_wex)
27:         E.insert(fdx + 1, aux)
28:     end for
29:     return update_method()
30: end function
31: function check_method()
32:     for (e ← 0, |E|) do
33:         for (i ← e, |E|) do
34:             if (E_{e[job]} = E_{i+1[job]}) then
35:                 src ← R[E_{e[agent]}]
36:                 dst ← R[E_{i+1[agent]}]
37:                 if (src ∈ K) then
38:                     K.remove(src)
39:                 end if
40:                 if (dst ∋ K) then
41:                     if (E_{i+1[agent]} ∈ B) and (i − e > 0) then
42:                         M ← append(src+"->"+dst+"*")
43:                         K ← append(dst)
44:                     else
45:                         M ← append(src+"->"+dst+" ")
46:                     end if
47:                     if (i − e > 0) then
48:                         src ← R[E_{i+1[agent]}]
49:                         dst ← R[E_{e+1[agent]}]
50:                         M ← append(src+"->"+dst+A[a])
51:                         a ← a + 1
52:                     end if
53:                 else
54:                     M ← append(src+"->"+dst+"S")
55:                     L ← append(E.index(E_{i+1}))
56:                 end if
57:                 break
58:             end if
59:         end for
60:     end for
61:     for (i ← 0, |M|) do
62:         print(M[i])
63:     end for
64:     print("overlap idx: ", L)
65:     if (|L| > 0) then
66:         return fix_overlay(L)
67:     else
68:         return E
69:     end if
70: end function
```

VII. EXPERIMENTAL EVALUATION

This section describes the experiments performed using the proposals discussed in this study. The experimental results are divided into four subsections. An Intel Core i5-8250U 1.60GHz computer with 8GB of RAM generated the original and readjusted production schedules. First, the conformity of the readjusted schedules with the respective operations

defined for the serializer and pruning filter was evaluated and then compared with the original schedule. Next, the ability of the overlay filter to resolve overlaps in the production schedules was evaluated. The last subsection presents the results of the readjusted schedules executed by MAS agents on the FMS simulation platform (see Section V). A video of the FMS operating with schedules readjusted by the three-stage filter and supplementary material with the experiment schedules and respective diagrams is available online[2].

A. EXPERIMENT 1

The first experiment verified the conformity of the serializer filter to production schedules susceptible to deadlocks. Fig. 9 shows the Gantt diagrams for the three production schedules with deadlocks generated by the FJS algorithm for comparison with the schedules readjusted by the serializer filter. The duration of events in the schedules was set to fixed-sized time units (5 s each) to make the diagrams easier to visualize.

The first schedule specifies the manufacturing of three different products whose order of event execution is numbered (Fig. 9-A). The schedule has deadlocks owing to task parallelism in null-buffered manufacturing resources (events 1 through 7, circled in the figure). The modified schedule, just below the initial schedule generated by the FJS, separates the blocking events from the parallel execution and sets them up for individual processing in the MAS. These events are assigned unique token numbers and organized in a new schedule to maintain the original sequence of manufacturing resources defined for the products. In Fig. 9-A, three events are removed from the parallel execution in the modified schedule (shown in the individual columns in the diagram). Serializing the events to avoid deadlocks in the FMS increased the completion time of the modified schedule by 5 s.

The second schedule specifies the manufacturing of two types of products (Fig. 9-B). However, this schedule causes deadlocks in tasks related to the DSI, DSC, and NFC resources, as circled in the figure. The serializer filter method alters the initial schedule created by the FJS by separating these events from the parallel execution. Serializing the blocking events increased the completion time of all tasks by 5 s in the new schedule. In the FMS studied, schedules must be executed repeatedly to reach the number of products defined in order. Therefore, the schedule completion time must be multiplied by the number of products required. For the modified schedule shown in Fig. 9-B, 20 units of each product type required a simulation time of 13 min.

Finally, the third schedule deals with three types of parts (Fig. 9-C), defined by $REF \prec DSI \prec NFC \prec MS1 \prec MS7 \prec DSO \prec REF$ for the first type, $REF \prec DSI \prec MPO \prec SLD \prec DSO \prec REF$ for the second type, and $REF \prec DSI \prec DSC \prec NFC \prec HBW \prec REF$ for the last type. In the first type, the VGR robot should move from its reference point (REF) to the part arrival location (DSI). The VGR robot picks up a part in the DSI and positions it in the NFC reader unit. Subsequently, the part should be transported for machining at the MS1 station. From MS1, the part is transported to another station (MS7), and after processing, it is deposited at the part removal point (DSO). Finally, the VGR robot returns to its reference point (REF), and the return from the VGR to the REF is implicit at the end of each product. The other two types of parts follow similar processes for machining and delivery or for stock in HBW. Because the schedule generated by the FJS algorithm was blocked, the proposed method serialized events that could not be executed in parallel to solve the null-buffer problem. Serializing events increased the task completion time for the modified schedule by 35 s.

Table 3 presents the details of the comparison among the five more extensive schedules, each with three types of parts. Each part type is specified by a sequence of production resources as in $REF \prec DSI \prec DSC \prec NFC \prec MS1 \prec MS4 \prec DSO \prec REF$ (Schedule 1). Each resource in the sequence was associated with an event with a token number. For the previously mentioned part types, the tokens of the events were numbered from 0 to 7 in the original schedule (FJS). In the readjusted schedule using a serializer filter (SRF), the token values were 0-3 and 9-12.

Table 3 lists the number of events, number of transitions per schedule, and the completion time for each. Except for the reference position (REF) for the three-axis VGR robot, the other items in the sequences were manufacturing features. Therefore, the number of events in the schedule is equal to the sum of the resources in all the sequences. For the schedules listed in the table, the number of events ranged from 24 to 27, whereas the number of transitions ranged from 41 to 45 for FJS and 29 to 38 for SRF. The completion time for all tasks is longer in the modified schedule (theoretically) owing to the serialization of the null buffer resources. However, unlike the FJS, the SRF guarantees that schedules are fully executed, considering the perfect functioning of resources in the FMS. In practice, if the FJS schedules are not blocking, they may demand more makespan than the readjusted schedules because of more transitions, which implies using the VGR robot.

For information purposes, the processing times of the FJS and serializer filter methods for generating the schedules listed in Table 3 were computed. These times were between 13.92 ms and 14.81 ms for the FJS and between 14.60 ms and 15.25 ms when using the serializer filter; that is, 0.68 ms for the worst time and 0.44 ms for the best time. On a more extensive schedule, with 11 products and 90 events, the schedule generation time was computed to be 42.81 ms for the FJS and 45.54 ms for the serializer filter.

B. EXPERIMENT 2

The second experiment verified the optimization capability of the pruning filter for the given production schedules, and its relevance as a readjustment method. Fig. 10 shows the

[2]3-*Stage Filter* video https://youtu.be/farVogeNT8M and supplementary material https://github.com/alexlds77/access2023.

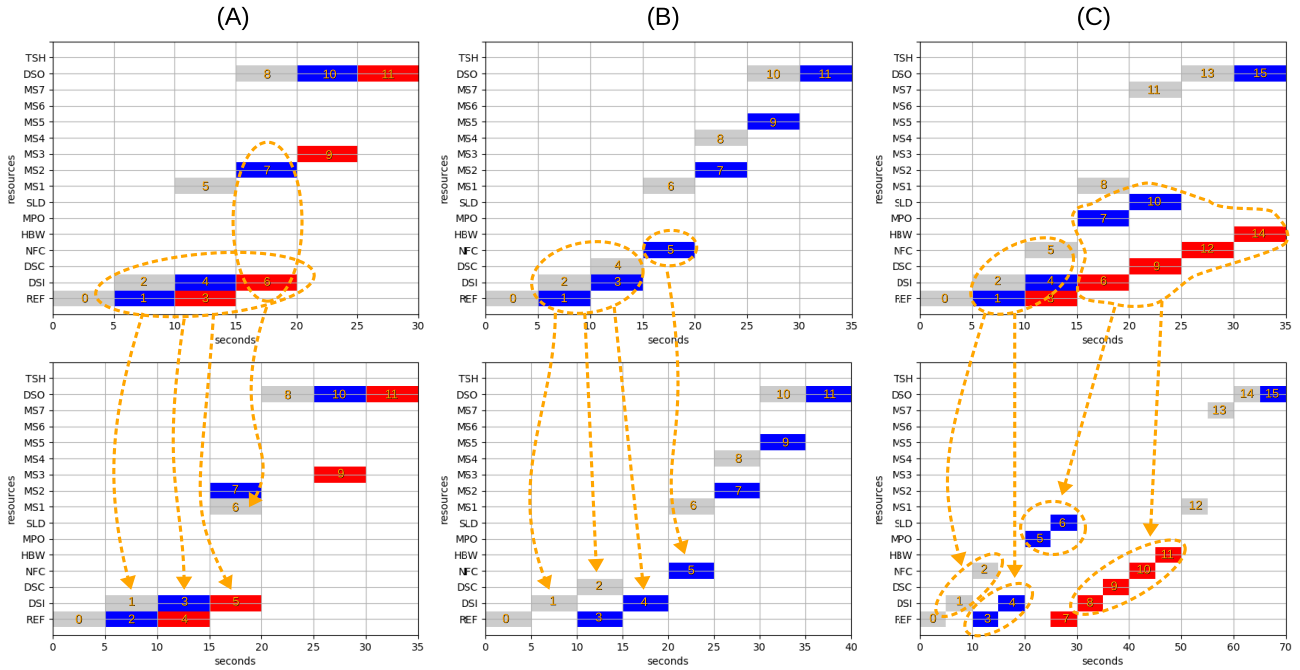

FIGURE 9. Serializer filter conformance certification.

TABLE 3. Comparison between FJS and SRF schedules.

Sched.	Manufacturing sequence (resources by product)	Tokens per event		Events	Transitions		Optimal length (s)	Modified length (s)
		FJS	SRF		FJS	SRF		
1	REF ≺ DSI ≺ DSC ≺ NFC ≺ MS1 ≺ MS4 ≺ DSO ≺ REF	0-7	0-3,9-12	24	41	32	✗ 45	✓ 70
	REF ≺ DSI ≺ DSC ≺ NFC ≺ MS2 ≺ MS5 ≺ DSO ≺ REF	1-8	3-6,9,10,12,13					
	REF ≺ DSI ≺ DSC ≺ NFC ≺ MS3 ≺ MS6 ≺ DSO ≺ REF	2-9	6-10,12-14					
2	REF ≺ DSI ≺ MPO ≺ SLD ≺ DSC ≺ MS2 ≺ NFC ≺ DSO ≺ REF	0-8	0-4,9,10,14,15	25	44	30	✗ 45	✓ 80
	REF ≺ DSI ≺ MS3 ≺ MS7 ≺ MS5 ≺ DSC ≺ NFC ≺ DSO ≺ REF	1-9	4,5,8-12,15,16					
	REF ≺ HBW ≺ MPO ≺ SLD ≺ MS5 ≺ HBW ≺ REF	2-8	5-8,12-14					
3	REF ≺ HBW ≺ MS1 ≺ DSC ≺ NFC ≺ MS3 ≺ DSC ≺ DSO ≺ REF	0-8	0,1,4-6,8,9,14,15	26	44	32	✗ 50	✓ 80
	REF ≺ DSI ≺ DSC ≺ MS7 ≺ DSC ≺ NFC ≺ HBW ≺ REF	1,2,4,5,7-10	1-3,9-13					
	REF ≺ HBW ≺ MS6 ≺ MPO ≺ SLD ≺ MS6 ≺ DSC ≺ DSO ≺ REF	2-10	3,4,6-8,13-16					
4	REF ≺ DSI ≺ NFC ≺ MS1 ≺ MS2 ≺ MS3 ≺ DSC ≺ DSO ≺ REF	0-8	0-2,4-7,9,10	27	45	38	✗ 55	✓ 65
	REF ≺ DSI ≺ MS2 ≺ DSC ≺ MS3 ≺ DSC ≺ MS7 ≺ DSC ≺ DSO ≺ REF	1-4,6-11	2-5,7,8,10-13					
	REF ≺ HBW ≺ MS4 ≺ MS5 ≺ MS6 ≺ MS7 ≺ DSO ≺ REF	2-9	3-5,7-9,11,12					
5	REF ≺ HBW ≺ DSC ≺ NFC ≺ MPO ≺ SLD ≺ DSO ≺ REF	0-7	0-5,14,15	25	43	29	✗ 45	✓ 85
	REF ≺ HBW ≺ MS5 ≺ NFC ≺ DSC ≺ MS2 ≺ MS3 ≺ DSO ≺ REF	1-9	5-9,14-17					
	REF ≺ DSI ≺ MS7 ≺ MS3 ≺ DSC ≺ NFC ≺ HBW ≺ REF	2-9	6,7,9-14					

Gantt diagrams for the three initial production schedules and compares them with the respective diagrams of the schedules readjusted by the pruning filter.

The schedules shown in Fig. 10-A specify the manufacturing of the three types of parts. The initial schedule (top) includes events with null buffer manufacturing resources that are serialized and contiguous with events with limited buffer or limited transient buffer resources. The circles and arrows in the respective diagrams highlight these events. The arrows are also labeled in the diagrams of the modified schedules, with "s" for events placed side by side and "p" for pruning. In the modified diagram shown in Fig. 10-A (bottom), the method breaks the contiguity by separating buffer-limited events while parallelizing and prioritizing the last event of each serialized sequence in the schedule.

The pruning filter reduces the length of the initial schedule to 115 s. The state diagram of the modified schedule in Fig. 10-A displays the sequence of events and their respective tokens. Events related to each part or product type are represented in a state diagram using colors (gray, blue, or red). The numbered arrows represent the movement of the parts according to the evolution of the tokens, whereas the dashed arrows represent the free movement (no parts) of the VGR robot. The DSI, DSC, NFC, MPO, and SLD resources were null-buffered in the factory, whereas the other resources were limited buffered, except for the reference position (REF). The MPO and SLD were considered null buffer resources because

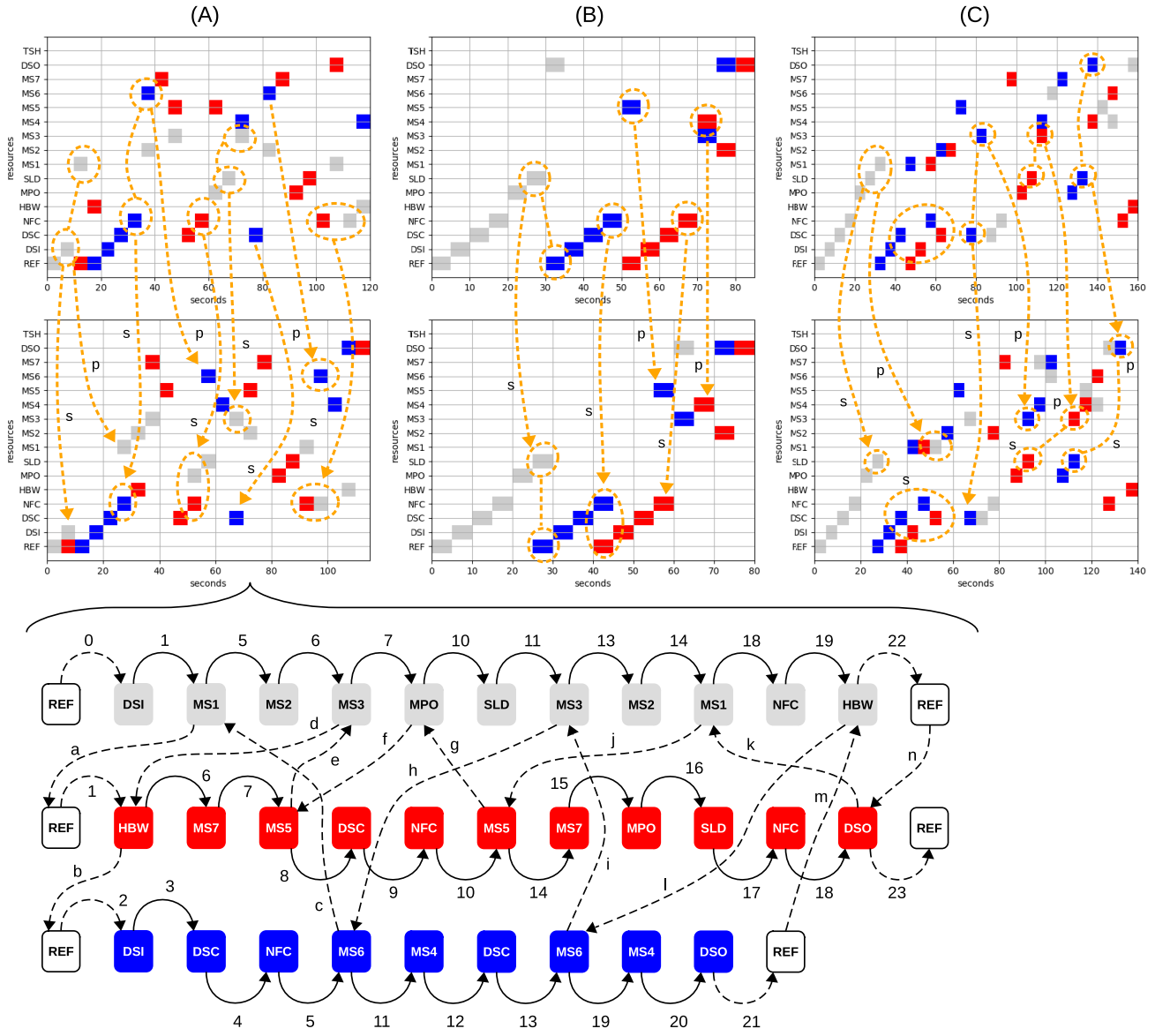

FIGURE 10. Pruning filter conformance certification.

the VGR robot did not reach the SLD. The SLD is accessed by the MPO, and from the moment a part is placed in the MPO, it is processed and sent to the SLD without dependency on the VGR robot. The number of tokens for the initial schedule is 26, whereas that for the readjusted schedule is 24. The MAS must process 37 events to produce three product types in the FMS.

The initial schedule in Fig. 10-B also has null buffer events contiguous with bounded buffer events. The diagram of the modified schedule identifies the parallelized events (labeller with "s") and the detached events (labeller with "p") from contiguity as defined by the pruning filter. This method reduced the initial schedule length to 80 s. The number of tokens for the initial schedule is 18, whereas that for the readjusted schedule is 16; each schedule has 24 events. Finally, Fig. 10-C shows a diagram of the initial and readjusted schedules using a pruning filter. The total number of events per schedule is 46. This method reduces the length of the initial schedule from 160 s (33 tokens) to 140 s (28 tokens).

C. EXPERIMENT 3

The third experiment highlighted the importance of the overlay filter in readjusting production schedules with an overlap. Figure 11 shows the Gantt diagram for the three schedules previously readjusted by serialization and pruning filters with deadlocks owing to overlap. Fig. 11-A (top diagram) shows a production schedule with three parts. $REF \prec DSI \prec MPO \prec SLD \prec DSC \prec MS3 \prec NFC \prec DSO \prec REF$ defines the production sequence for the first part (in blue color),

$REF \prec DSI \prec NFC \prec MS1 \prec MS6 \prec MS4 \prec DSC \prec DSO \prec REF$ is the sequence for the second type (red part), and $REF \prec HBW \prec MPO \prec SLD \prec MS4 \prec HBW \prec REF$ for the third type (gray). However, the schedule will cause deadlock owing to overlap when the VGR robot moves the gray part from resource SLD to resource MS4 (transition "a") and then tries to move the red part from MS1 to MS6 and MS4 (transitions "b" and "c"). Deadlock occurs because of the circular wait for the VGR robot when the MS4 process (identified by 18 in the diagram) waits for the robot to remove the gray part. Simultaneously, process MS6 (identified as 15), which occupies the robot, waits for MS4 to become available. The overlay filter solves the overlap problem by switching the order of events in the schedule, as shown in Fig. 11-A a (bottom diagram). First, the VGR robot would move the gray piece from *SLD* to *MS*4 (transition "a"), and then it would move the red part from *MS*1 to *MS*6 (transition "b"). Upon completing the previous tasks, the robot would move the gray part from *MS*4 to *HBW*, leaving *MS*4 free to process the red part (transition "d"). The overlap filter performs a single cycle of readjustments to leave the schedule free of overlaps. The theoretical completion time of all tasks for the readjusted schedule increased by 5 s compared to the initial schedule.

Fig. 11-B (top diagram) shows another production schedule for three part types defined by sequences $REF \prec HBW \prec MPO \prec SLD \prec MS3 \prec MS5 \prec NFC \prec REF$ for the first part type (gray color), $REF \prec DSI \prec MPO \prec SLD \prec DSC \prec MS5 \prec MS2 \prec NFC \prec HBW \prec REF$ for the second type (blue part), and $REF \prec DSI \prec NFC \prec MS1 \prec MS4 \prec MS3 \prec DSC \prec DSO \prec REF$ for the third type (red part). The schedule will cause deadlock owing to overlap at *MS*5 when the VGR robot moves the blue part from resource *DSC* to resource *MS*5 (transition "a") and then tries to move the gray part from *MS*3 to *MS*5, which will already be occupied (transition "b"). By the schedule, transition "a" will occur first, then transition "b." However, transition "d," which would release *MS*5, will only happen after transition "c," resulting in circular waiting. The overlay filter solves the deadlock problem as highlighted in Fig. 11-B (bottom diagram), with the VGR robot prioritizing the transport of the blue part from *NFC* to *MS*5 (transition "a") and from *MS*5 to *MS*2 (transition "b"). With the *MS*5 free, the agent coordinating the VGR robot can deposit the other part (gray color) in the resource (transition "c"). Therefore, the readjusted schedule is free of overlays to be executed by the MAS. The overlay filter also performs a single readjustment cycle on the original schedule to solve the overlapping problem.

However, to solve the overlap problem in *MS*3, as highlighted in the schedule in Fig. 11-C (top), the overlay filter performed two cycles of readjustments (recursively). The original schedule specifies three part types, and the overlay is predicted with the movement of the gray part from resource *MS*2 to resource *MS*3 (transition "b"), whereby *MS*3 is already occupied by the red part (transition "a," from *DSI* to *MS*3). However, the method avoids deadlock by prioritizing the removal of the red part from *MS*3 (transition "b"). Thus, *MS*3 will be free to process the gray part (transition "c"), as displayed in the diagram for the readjusted production schedule (Fig. 11-C, bottom). The theoretical task completion times for the readjusted schedules in Fig. 11-B and Fig. 11-C increased by 5 s. This time increase refers to prioritizing an overlapping event with a shared production resource. Therefore, overlapping eventually occurs only in schedules with shared production resources between two or more part types. Event prioritization is achieved with serialization. In the worst case, the overlay filter serializes all events of each part type recursively to avoid overlap problems. However, this method combines serializer and pruning filters for deadlock resolution by null buffering and contiguity breaking for schedule optimization.

Table 4 presents data comparing other production schedules initially readjusted by the serializer and pruning filters (S+P) with the readjustments performed by the overlay filter (OVF). The first schedule in the table (schedule 1) specifies three part types according to production sequences and the respective part color indicated at the beginning of each sequence. The schedule consists of 24 events, and the number of transitions is 38 for the initial schedule (S+P) and 30 for the schedule modified by the overlay filter (OVF). The method performed five cycles of readjustments to solve the deadlock problems.

In the first cycle, the method predicts overlaps in resources *MS*1, *MS*3, and *MS*7 and readjusts the schedule. The method identified an overlap in *MS*5 (cycle 2) and *MS*7 (cycle 3). In the fourth cycle, the method solves the overlapping problems and obtains deadlock-free (ND) scheduling. The serializer and pruning filters were then run to ensure the serialization of the null buffer events and schedule optimization. Finally, the overlay checks that the schedule remains free of overlaps (cycle 5) because of possible readjustments by the serializer and pruning filters and confirms that no other deadlocks are found. The completion time of the tasks in the schedule, readjusted using the overlay filter, was 70 s.

The other schedules in Table 4 also specify the manufacturing of the three distinct part types. The number of events in the schedules varied between 19 and 30 and the number of transitions with readjustments performed by the overlay filter was smaller than that in the initial schedule. This method identified overlaps and readjusted all schedules to resolve deadlocks. In the best case, a schedule with an overlap requires three cycles of readjustments to be free of deadlocks (e.g., schedule 2). The overlap filter also solves the overlap problem for other cases; however, the readjustment process involves shifting the position of the null buffer events. The serializer and pruning filters then change the initial solution if it has deadlocks, and the overlay filter is rerun to resolve other deadlocks (e.g., within Schedules 5 and 6). The schedules readjusted by the overlay filter have a slightly longer optimal length than the initial schedules but can be executed deadlock-free by the MAS.

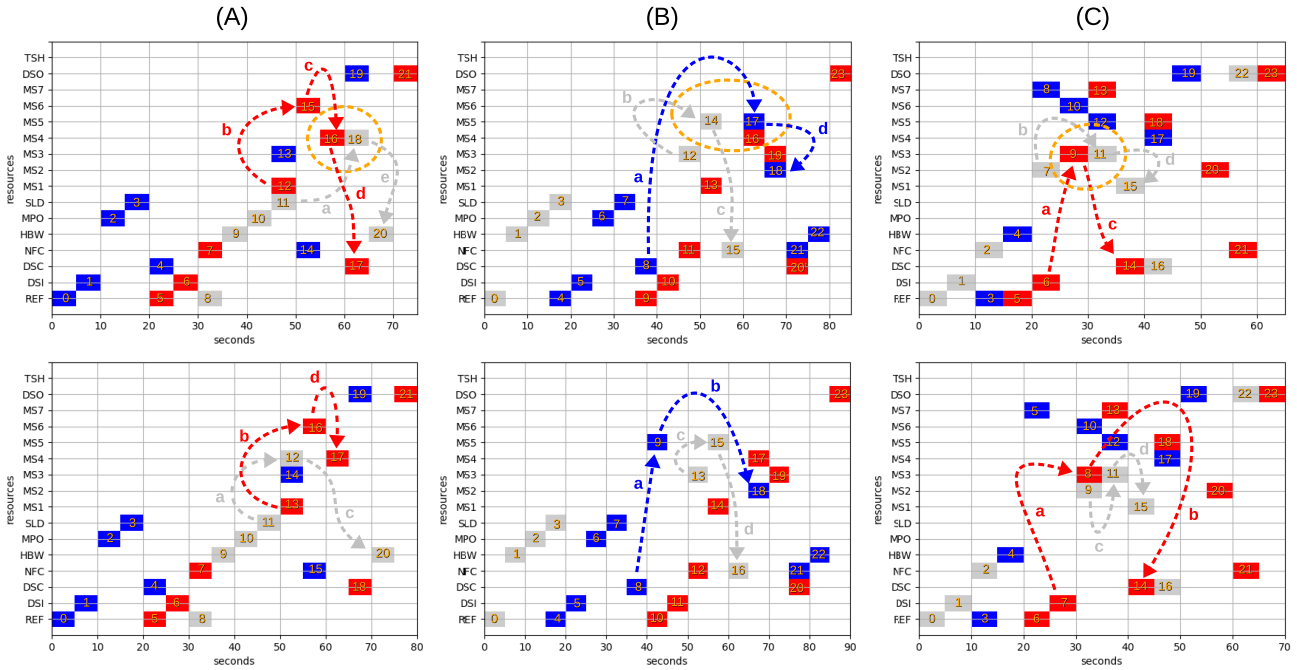

FIGURE 11. Overlay filter conformance certification.

TABLE 4. Comparison between SRF + PRF and OVF schedules.

Sched.	Manufacturing sequence (resources by product)	Events	Transitions		Deadlocks per cycle	Optimal length (s)	
			S+P	OVF		S+P	OVF
1	REF ≺ DSI ≺ MS1 ≺ MS3 ≺ MS5 ≺ MS7 ≺ DSO ≺ REF REF ≺ DSI ≺ MS1 ≺ MS3 ≺ MS4 ≺ MS6 ≺ DSO ≺ REF REF ≺ DSI ≺ MS7 ≺ MS5 ≺ MS2 ≺ MS4 ≺ DSO ≺ REF	24	38	30	1: MS1, MS3, MS7; 2: MS5; 3: MS7; 4: ND (✓); SRF + PRF; 5: ND (✓)	✗ 50	✓ 70
2	REF ≺ HBW ≺ MPO ≺ SLD ≺ MS1 ≺ DSC ≺ MS2 ≺ MS7 ≺ NFC ≺ HBW ≺ REF REF ≺ DSI ≺ MS6 ≺ DSC ≺ MPO ≺ SLD ≺ MS3 ≺ MS5 ≺ DSO ≺ REF REF ≺ DSI ≺ DSC ≺ MS1 ≺ MS5 ≺ MS4 ≺ DSC ≺ DSO ≺ REF	30	39	38	1: MS1, MS5; 2: (✓) ND; SRF + PRF; 3: (✓) ND	✗ 90	✓ 95
3	REF ≺ DSI ≺ MS2 ≺ MS1 ≺ DSO ≺ REF REF ≺ DSI ≺ DSC ≺ MS1 ≺ MS7 ≺ MS3 ≺ DSO ≺ REF REF ≺ HBW ≺ MS3 ≺ MS5 ≺ DSO ≺ REF	19	26	22	1: MS1, MS3; 2: MS3; 3: (✓) ND; SRF + PRF; 4: (✓) ND	✗ 50	✓ 60
4	REF ≺ DSI ≺ MPO ≺ SLD ≺ MS1 ≺ MS2 ≺ MS3 ≺ DSO ≺ REF REF ≺ DSI ≺ MS1 ≺ MPO ≺ SLD ≺ MS6 ≺ DSO ≺ REF REF ≺ HBW ≺ DSC ≺ MS2 ≺ MS5 ≺ DSC ≺ MS2 ≺ MS3 ≺ DSO ≺ REF	27	37	31	1: MS1, MS2; 2: MS2, MS3; 3: (✓) ND; SRF + PRF; 4: (✓) ND	✗ 75	✓ 95
5	REF ≺ HBW ≺ MPO ≺ SLD ≺ MS7 ≺ MS3 ≺ MS4 ≺ DSC ≺ HBW ≺ REF REF ≺ DSI ≺ DSC ≺ MS1 ≺ MS7 ≺ MPO ≺ SLD ≺ DSO ≺ REF REF ≺ DSI ≺ MS1 ≺ MS6 ≺ MS3 ≺ DSC ≺ DSO ≺ REF	27	35	35	1: MS7; 2: (✓) ND; SRF + PRF; 3: MS7; 4: (✓) ND	✗ 80	✓ 85
6	REF ≺ DSI ≺ MS1 ≺ MS6 ≺ DSC ≺ MS6 ≺ MS3 ≺ DSO ≺ REF REF ≺ DSI ≺ MS1 ≺ DSC ≺ MS3 ≺ MS5 ≺ DSO ≺ REF REF ≺ DSI ≺ MS7 ≺ MS6 ≺ DSO ≺ REF	26	35	31	1: MS7, MS1; 2: MS5; 3: (✓) ND; SRF + PRF; 4: MS1; 5: (✓) ND	✗ 70	✓ 85

D. EXPERIMENT 4

This subsection evaluates the performance of MAS in executing production tasks on a simulation platform. The two figures compare the operations performed by the agents with the readjusted schedules, and the table presents data on other production schedules used in the FMS. The results of the experiments are presented in detail along with the time spent on the operations (makespan). The times were obtained by measuring the MAS code execution times for each schedule.

Fig. 12 shows the execution of the production schedule with two types of parts. The first type is defined as $REF \prec DSI \prec MS1 \prec DSO \prec REF$, whereas the second type is defined as $REF \prec DSI \prec MS2 \prec DSO \prec REF$. Fig. 12-A, in the upper part of, shows the deadlock problem in the FMS during the execution of the original predictive schedule (FJS) without readjustment. The bottom of the figure shows the corresponding state diagram of the schedule in question. The deadlock problem occurs because DSI is a "null buffer" type resource and cannot be used for temporary part storage, thus preventing the VGR robot from being free to attend to other parallel tasks defined in the schedule. To avoid overlapping parts on the same resource, transitions "a" and "b" should not occur in the order the FJS algorithm defined them. Therefore, the schedule in

Fig. 12-A is blocking and can only be executed with due readjustment.

When considering the same schedule readjusted by the serializer filter, the execution of events occurs in a different order from that defined in the original schedule. This method serializes null buffer events, ensuring that other events dependent on the VGR robot do not execute in parallel with the null buffer events. Fig. 12-B (top) shows that the first part (in white) leaves the DSI (token 1) and is deposited in MS1 for processing (for 40 s). The MS1 event precedes the DSI event in the schedule. In this case, the VGR robot moves the current part back to the resource for the next event (token 2): the DSO. In parallel to the DSO event, transition ''a'' occurs, where the robot from the VGR goes to REF. The robot then picks up a new part (in blue) at the DSI and takes it to MS2 (tokens 2 and 3), as shown in the state diagram in the respective figure. At MS2, the part was processed (for 30 s) and then deposited in the DSO (token 4). In parallel to the DSO event, transitions ''b'' and ''c'' occur, where the VGR robot goes to REF, and the schedule ends. Events that occur in parallel in the schedule are identified using repeated number tokens (tokens 2 and 4, in this case). The makespan of the readjusted schedule with the serializer filter was 152.91 s.

The order of event execution in the schedule readjusted by the pruning filter follows the premise of maintaining the serialization of the null buffer events, parallelizing the last event in the serialized group, and prioritizing its execution for the VGR robot. Fig. 12-C shows the manner in which the filter operates at the top. After the first part (in white) is deposited at MS1, the VGR robot moves to pick up the other part (part in blue) at DSI (transition ''a''). Meanwhile, MS1 continues the processing (for 40 s). The new part is transferred from the DSI to MS2 (token 2) and deposited in the processing resource (for 30 s). The VGR robot then moves to the MS1 resource (transition ''b''). After the MS1 processing time elapsed, the first part was sent to the DSO (token 2). The processing of the production schedule continues according to the state diagrams. A crucial aspect of readjustment with the pruning filter is determining the pruning event (MS1 and MS2). Unlike the serializer, the pruning events occur in parallel, preventing the VGR robot from being idle, waiting 40 s for the completion of MS1 and another 30 s for the fulfillment of MS2. With the readjustment of the pruning filter, the makespan was reduced to 105.16 s compared to that of the serializer filter. The schedules in Fig. 12 are free of overlap and were successfully executed by the MAS.

Fig. 13 shows the execution of a production schedule with three types of parts. The first part type was defined as $REF \prec DSI \prec DSC \prec NFC \prec MS7 \prec DSO \prec REF$, the second as $REF \prec DSI \prec MPO \prec SLD \prec NFC \prec DSO \prec REF$, and the third as $REF \prec HBW \prec MS3 \prec DSO \prec REF$. The top panel of Fig. 13 shows the operations in the FMS, with arrows labeled with token numbers and transitions for each modified schedule. The yellow arrows represent the flow executed by the agents with a schedule readjusted using the serializer filter (SRF). In contrast, the arrows in cyan represent the flow executed by the pruning filter (PRF). Both schedules were subjected to the overlap filter and therefore did not overlap.

Fig. 13-A shows a state diagram of the serializer filter. There is a transition ''a'' when the schedule execution flow for the first part type reaches DSO (token 4). Subsequently, a ''context switch'' (OS analogy) occurs at the moment when the VGR robot stops moving the first part type and starts moving the new part (in blue). The execution flow with the robot moving the new part continues until it reaches the DSO-related event (Token 8). Next, with transition ''b,'' the robot moves to HBW (token 9), MS3 (token 10), and DSO (token 11). Finally, with the transition ''c,'' the VGR robot returns to the reference position (REF). The makespan of the readjusted schedule with the serializer filter was 304.9 s.

Fig. 13-B displays a state diagram of the pruning filter. In the execution flow for the first part type, the transition ''a'' occurs in MS7, not in DSO, because the event associated with MS7 is pruned from contiguity with the serialized null buffer events. Furthermore, the last event in the serialized group (NFC) is prioritized for execution over other events of the same token. The transition from the NFC to MS7 (token 3) occurs first, and then from REF to DSI. Thus, the part is left in MS7, which immediately starts its processing (for 20 s), whereas the VGR robot goes to REF (transition ''a'') and then to DSI (token 3). The flow occurs similarly to the serializer filter, except for DSO (token 9, transition ''c''), which directs the robot to remove the part from MS7 and place it in DSO. Finally, token 10 chains the robot to return to its reference position (REF). The makespan of the readjusted schedule with the pruning filter was 294.2 s.

Table 5 presents data from other schedules tested at the FMS under study. These schedules have also been certified by the overlay filter and are free of overlap. The first schedule refers to manufacturing two types of parts according to the color indicated at the beginning of the sequence of events for each part. The order of execution of the events for every kind of schedule is shown in the columns referring to the token. For schedule 1, the FJS algorithm defines resource mapping by tokens as $REF(0) \prec DSI(1) \prec MPO(2) \prec SLD(3) \prec MS7(4) \prec MS4(5) \prec HBW(7) \prec REF(8)$ for the white-colored part and $REF(1) \prec DSI(2) \prec MS1(3) \prec MS3(5) \prec MS2(6) \prec DSO(8) \prec REF(9)$ for the blue-colored part. Therefore, if the original FJS schedule were not blocking, the MAS agents would execute the events in the order REF (token 0), DSI and REF (token 1), MPO and DSI (token 2), SLD and MS1 (token 3), until they reached the last REF-related event (token 9).

For schedule 1, readjusted by the serializer filter, the mapping is $REF(0) \prec DSI(1) \prec MPO(2) \prec SLD(3) \prec MS7(4) \prec MS4(7) \prec HBW(8) \prec REF(9)$ for the white-colored part and $REF(4) \prec DSI(5) \prec MS1(6) \prec MS3(7) \prec MS2(8) \prec DSO(9) \prec REF(10)$ for the blue-colored part. Therefore, MAS agents execute schedules differently, avoiding deadlocks that occur in the original schedule. Similarly, the mapping and execution orders of the same schedule readjusted by the pruning filter differed.

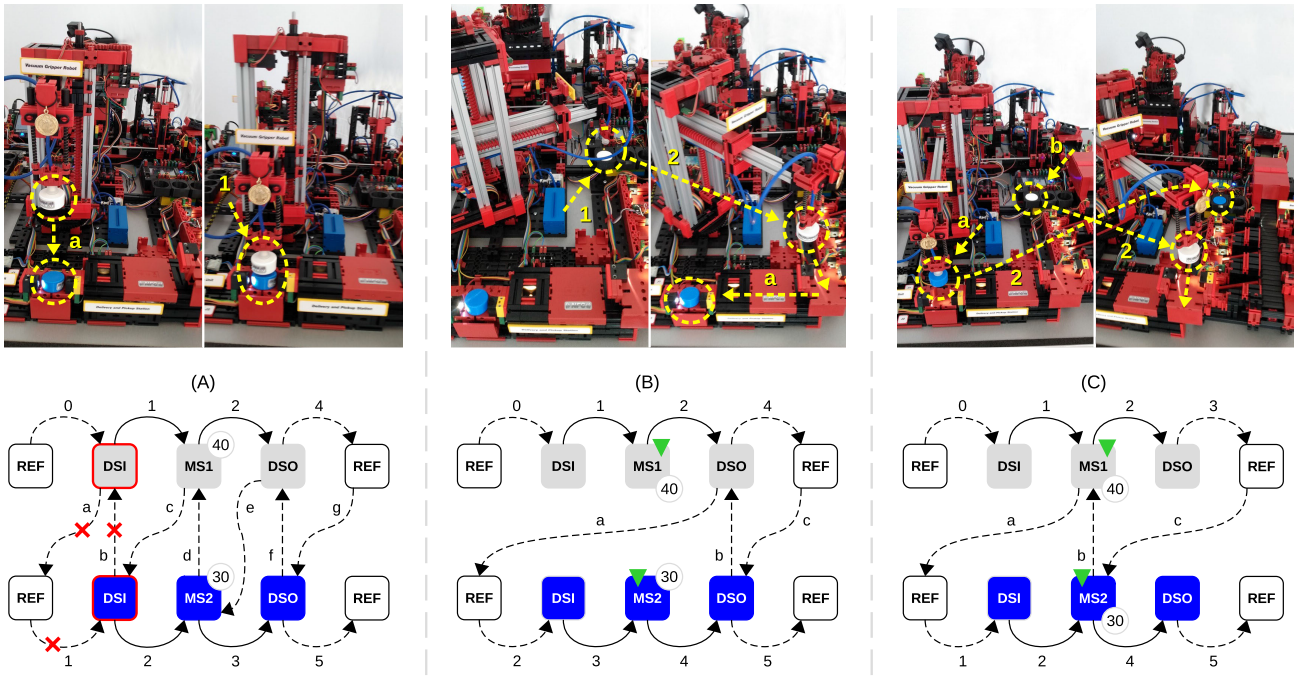

FIGURE 12. Operations performed in the SRF and PRF schedules.

TABLE 5. Comparison between FJS, SRF and PRF schedules.

Sched.	Manufacturing sequence (resources by product)	Tokens per event			Events	Transitions			Makespan (s)		
		FJS	SRF	PRF		FJS	SRF	PRF	FJS	SRF	PRF
1	REF ≺ DSI ≺ MPO ≺ SLD ≺ MS7 ≺ MS4 ≺ HBW ≺ REF	0-5,7,8	0-4,7-9	0-3,6-9	15	24	18	18	✗ 0.00	✓ 331,22	✓ 324,67
	REF ≺ DSI ≺ MS1 ≺ MS3 ≺ MS2 ≺ DSO ≺ REF	1-3,5,6,8,9	4-10	3-7,9,10							
2	REF ≺ DSI ≺ DSC ≺ NFC ≺ MS6 ≺ HBW ≺ REF	0,1,3,4,6,10,12	0-4,11,13	0-3,9-11	19	32	23	23	✗ 0.00	✓ 301.41	✓ 285.43
	REF ≺ DSI ≺ NFC ≺ MPO ≺ SLD ≺ DSO ≺ REF	1,3,7,9,10,13,14	4-9,14	3-7,12,13							
	REF ≺ HBW ≺ MS1 ≺ DSO ≺ REF	2,5,8,11,13	9-13	7-10,12							
3	REF ≺ DSI ≺ MS1 ≺ DSO ≺ REF	1,3,4,9,10	2-4,8,10	1,2,6-8	15	23	20	20	✗ 0.00	✓ 212.66	✓ 144.63
	REF ≺ DSI ≺ MS2 ≺ DSO ≺ REF	0,1,3,6,7	0-2,6,9	0,1,3,4,7							
	REF ≺ DSI ≺ MS3 ≺ DSO ≺ REF	2,4,5,8,9	4-8	2-6							
4	REF ≺ DSI ≺ DSC ≺ NFC ≺ MS7 ≺ DSO ≺ REF	1,3-5,7,11,12	2-6,13,14	1-4,10-12	21	35	24	24	✗ 0.00	✓ 338.79	✓ 333.64
	REF ≺ DSI ≺ DSC ≺ NFC ≺ MS7 ≺ DSO ≺ REF	2,5,8-10,12,14	6-11,14	4-8,12,13							
	REF ≺ DSI ≺ MS3 ≺ MS2 ≺ NFC ≺ HBW ≺ REF	0,1,3,6,10,13,15	0-2,11-13,15	0,1,8-10,13,14							
5	REF ≺ DSI ≺ DSC ≺ NFC ≺ MS1 ≺ MS2 ≺ MS3 ≺ DSO ≺ REF	0,1,3-5,7,9,12,15	0-4,8,10,11,13	0-3,5-7,9,11	21	36	29	29	✗ 0.00	✓ 271.98	✓ 264.52
	REF ≺ DSI ≺ MS6 ≺ MS4 ≺ DSO ≺ REF	2,6,11,13,16,17	6-8,12,14,15	4,5,9,10,12,13							
	REF ≺ DSI ≺ MS5 ≺ MS7 ≺ HBW ≺ REF	1,3,6,8,10,14	4-6,9,10,12	3,4,6-8,10							
6	REF ≺ DSI ≺ MS7 ≺ DSO ≺ REF	0-2,6,7	0-2,6,7	0,1,5-7	10	14	10	10	✗ 0.00	✓ 139.54	✓ 117.49
	REF ≺ DSI ≺ MS3 ≺ DSO ≺ REF	1-5	2-6	1-5							
7	REF ≺ DSI ≺ MS1 ≺ MS2 ≺ MS1 ≺ MS2 ≺ DSO ≺ REF	0-6,8	0-2,8-10,12,14	0,1,5-8,10,12	21	37	28	28	✗ 0.00	✓ 348.45	✓ 335.64
	REF ≺ DSI ≺ MS3 ≺ MS4 ≺ MS5 ≺ HBW ≺ REF	1-7	2-4,9,11-13	1,2,7-11							
	REF ≺ DSI ≺ MPO ≺ SLD ≺ HBW ≺ REF	2-5,7,9	4-8,14	2-5,12,13							
8	REF ≺ DSI ≺ DSC ≺ NFC ≺ MS5 ≺ MS7 ≺ HBW ≺ REF	2-6,8,10,11	7-12,14,15	5-8,10-13	24	44	31	31	✗ 0.00	✓ 402.78	✓ 386.25
	REF ≺ DSI ≺ MPO ≺ SLD ≺ MS6 ≺ HBW ≺ REF	0-2,8,10,12,14	0-4,16,17	0-3,14-16							
	REF ≺ DSI ≺ DSC ≺ MS3 ≺ MS2 ≺ MS1 ≺ MS4 ≺ DSO ≺ REF	1-4,6,7,9,11,13	4-7,11-13,15,17	3-5,8-11,13,15							

Therefore, the makespan values are different: 331.2 s for the serializer filter and 324.6 s for the pruning filter. The number of events in the readjusted schedule was the same as that in the original schedule. However, with the serialization of some events, the number of transitions was lower in the readjusted schedules (18 transitions). In this study, the number of transitions is equal to the number of movements performed by the VGR robot to transport the parts in the FMS to the completion of the schedule.

In Schedule 2, the numbers of parts (3), events (19), and transitions (32, 23, and 23) were higher than those in the previous schedule. However, the makespan times were shorter, which means that the distance between the resources traveled by the VGR robot is shorter (the resources are close to each other), or the processing time of the parts on the resources is shorter. The makespan obtained by running the SRF-modified schedule was 301.4 s, whereas that of the PRF-modified was 285.4 s.

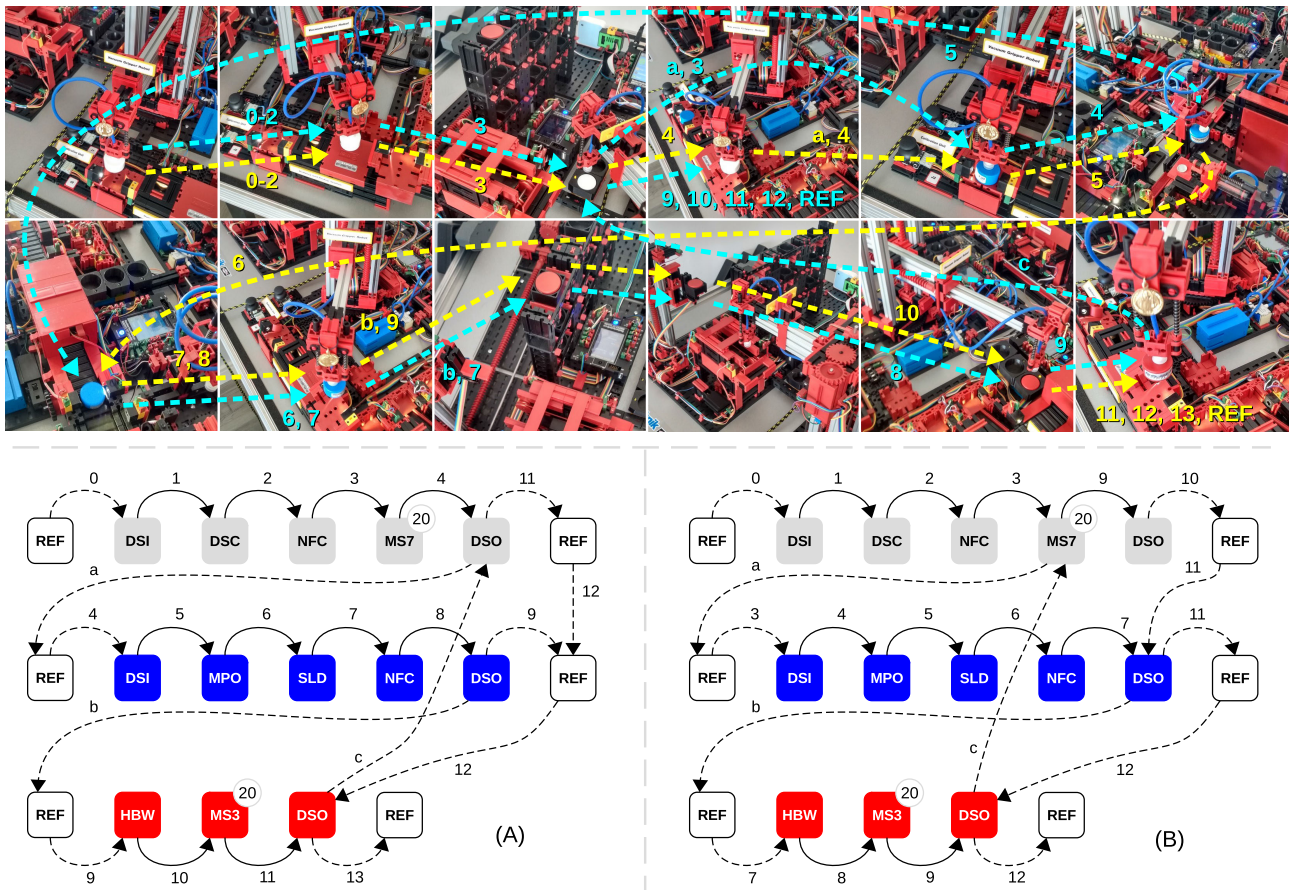

FIGURE 13. Operations performed in the SRF and PRF schedules.

Schedules 3-8 followed the same pattern as the makespan obtained using the proposed filters. In all these schedules, the pruning filter had a lower makespan than the serializer because of pruning, which avoided the idleness of the RGR robot and because of the parallelism and prioritization of certain events with null buffer resources. Notably, a smaller makespan value for a pruning filter is not a rule. For example, in schedule 6, station MS7 processes a part for 30 s, and MS3 for 10 s. If the processing time at MS7 and MS3 is set to 5 s the makespan of the serializer becomes smaller than that of pruning, respectively 100.04 s and 109.97 s. Owing to the time issue, this situation occurs because it is more advantageous for the VGR robot to wait to complete the part at MS7 rather than move to other resources and then return to MS7.

VIII. OVERALL EVALUATION

The proposed three-stage readjustment filter for predictive schedules successfully avoids specific deadlock problems in the FMS under study. The results showed that the number of events in the readjusted schedules equaled that in the original schedule. Even with the serialization of some events, the makespan obtained was similar to the optimal length calculated by the predictive algorithm because the number of transitions in the readjusted schedules was smaller. This condition implies that no additional events require more processing time from the system, no extra buffers are required, and no other actions imply rescheduling at runtime. Although significant research exists for dealing with deadlock problems in production scheduling, such as [12], [13], [16], [17], [24], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35], [36], [38], [39], [40], [41], [42], [43], and [44], approaches aimed at dynamic scheduling and rescheduling for runtime deadlock recovery rely on information about the system state that will only be available during its execution.

However, the readjustment focuses on defining the order of events and transitions to ensure a functional (deadlock-free) production schedule. Furthermore, the consensus control of the proposed MAS architecture provides synchronization at predefined periods, allowing agents to deduce the system's current state according to the percentage of events executed at each token change, which is interesting for focused research in rescheduling. Consensus control is crucial for agents to cooperate in the execution of manufacturing tasks defined by scheduling. Synchronism ensures a predictable response to demand and production stability. The communication layer

and consensus mechanism are the same across all agents, favoring the scalability and adaptability of the MAS architecture to other scenarios. The scalability of the architecture was attested using virtual agents developed for the FMS under study for informational purposes.

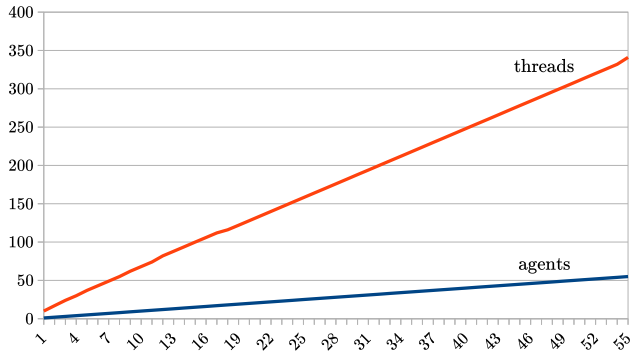

FIGURE 14. Scalability of the MAS architecture.

Fig. 14 shows the load scalability in the relationship between the agents and threads for the agent model presented in Section IV. On an average, a single agent in the system ran between six and seven threads. With two agents, each ran 13-14 threads; with three agents, 20 threads each; and so on. Fifty-five virtual agents were run on the computer described in Section VII, resulting in 341 threads per agent, or 18755 threads overall. As the number of agents in the system increases, the number of threads increases exponentially owing to the increased connections between agents. Because the agents are implemented with threads for each ROS topic and its control functions, the total number of agents in the MAS is limited only by the resources of the computer hosting the virtual agents. However, in a real scenario, the computational load is distributed on the hardware of the production resources, and this limitation is irrelevant. In this case, the bandwidth is likely to be a limiting resource. But there are a variety of solutions that allow the integration of ROS-based systems in industrial environments, such as the ROS-Industrial [48] software package that contains libraries, tools, and drivers for industrial hardware. In addition, there are converters for integration into industrial networks (e.g., PROFINET networks) and gateways that allow quick and easy access to industrial IoT ecosystems.

The proposals of this study ensures that different production schedules, defined based on a complete view of the FMS state, are executed deadlock-free. The MAS replaces the centralized fixed-cycle production control of the original platform. The platform was adapted to FMS concepts to ensure production flexibility using the proposed approaches. Furthermore, the MAS agents can run simple predictive schedules for a single product at a time, in any manufacturing sequence, or with repeated resources. In this case, the readjustment filters are dispensed because deadlocks do not occur in the system. Therefore, considering all the part-processing resources available in the FMS under study, the system reconfiguration flexibility would theoretically admit up to 14. (Factorial) Simple Predictive Schedule. Other situations could be addressed by including new readjustment methods. For example, a filter that considers the relationship between the path and processing time prioritizes schedules that can be executed in less time.

Replicating the approach for other scenarios is straightforward but requires some adaptations. For readjustment filters, it is necessary to indicate the null-buffer resources for the serializer filter (Alg. 1, variable B) and the resources that the pruning filter can prune (Alg. 2, variable D). For the overlay filter, it is necessary to indicate the resources that can serve as a limited buffer and the nomenclature of all the FMS resources (Alg. 3 for variables B and R, respectively). Production schedules should be made available for the filters according to the model defined in subsection IV-A. For the MAS architecture, the communication layer and consensus can be replicated without adaptations with ROS-based agents. The most significant adaptation work is concentrated on the control and physical layers because they are specific to the manufacturing resources of the new scenario. Some available solutions, such as OpenPLC [49], an open-source Programmable Logic Controller compatible with Windows and Linux host platforms, can facilitate this integration, such as the Raspberry Pi and derivatives (e.g., UniPi and PiExtend). Any controller with embedded Linux, supporting Python and rospy [50] (Python client library for ROS), can be used as the host system for the agents. For example, the FMS in this research employs ARM Cortex A8 (32-bit/600MHz) + Cortex M3 controllers and embedded Linux, compiled with a tool called Buildroot[3]. With this issue resolved, redirecting the event-processing flow would be sufficient for the control to perform manufacturing tasks in the new scenario.

TABLE 6. Efficiency of *Readjustment 3-Stage Filter*.

	Sched.	Completed schedules			Efficiency (%)		
		FJS	S+P	OVF	FJS	S+P	OVF
Expt.1	8	0	6	8	✗ 0	✓ 75	✓ 100
Expt.2	3	0	3	3	✗ 0	✓ 100	✓ 100
Expt.3	9	0	0	9	✗ 0	✗ 0	✓ 100
Expt.4	10	0	10	10	✗ 0	✓ 100	✓ 100
				AVG.	0.00	68.75	100.00

Table 6 summarizes the data on the number of completed schedules and efficiency of the proposed methods for each experiment in Section VII. The experiments included 30 schedules, considering all figures and tables in the section. The last row of the table presents the sum of the completed schedules for each algorithm and their respective efficiency percentages. In the first experiment (Expt.1), the serializer filter (SRF) was 75%. In comparison, the FJS is 0% for all the eight production schedules used in the experiment. A percentage of 0% indicates that no FJS schedule can be completed in

[3]Buildroot is available at https://buildroot.org/

the FMS because of deadlocks. In the other experiments, the percentage of 0% for the FJS schedules was repeated because the algorithm did not consider production resource sharing or transportation system limitations when creating schedules. In the third experiment (Expt.3), the efficiency of the serializer and pruning filters was 0% because the readjusted schedules contained deadlocks by overlapping (ditto for two schedules from Expt.1). Finally, the combination of the three-stage filter represented in the table by OVF showed 100% efficiency for all sets of schedules evaluated.

A. COMPLEXITY OF ALGORITHMS

Information regarding the time complexities of the proposed algorithms is essential for evaluating the employment of these approaches in other production environments. Time complexity represents the time required for an algorithm to execute a function of an input length. Table 7 presents data on the time complexity of the developed agents and readjustment filters. The mathematical notation Big–O was used to calculate the time complexity for the best and worst cases.

TABLE 7. Time complexity of the proposed algorithms.

		Best	Worst
Agents	Signaling	$O(n + 7)$	$O(n^2 + 3n + 13)$
	Synchrony	$O(n + 16)$	$O(n^2 + 5n + 21)$
	Notification	$O(5)$	$O(n + 7)$
	Broadcast	$O(2)$	$O(2)$
Filters	Serializer	$O(2n^2 + 10n + 28)$	$O(2n^2 + 14n + 46)$
	Pruning	$O(n^2 + 4n + 27)$	$O(n^2 + 8n + 33)$
	Overlay	$O(2n^2 + 9n + 52)$	$O(2n^2 + 12n + 61)$

The time complexity of the agents refers to a single agent, because the proposed model is standard for all MAS agents. According to the model (subsection IV-C), the complexity calculation focuses on the communication layer of the MAS architecture. In comparison, the control and physical layers are specific to the resources associated with each agent; therefore, they are omitted from the calculation.

Conditional statements, assignments, and simple operations are computed at constant times. Other more complex operations were calculated according to the time complexity defined in the documents on the current CPython. The sending and receiving of messages in code blocks for consensus (see Fig. 3) were computed with a constant time $O(1)$. In Table 7, the sum of the complexities calculated for ''signaling,'' ''synchrony,'' and ''notification'' represents the time complexity of the consensus. Finally, the temporal complexities of the serializer, pruning, and overlay filters are presented. In the overlay filter, the temporal complexity of the recursion is not computed, because it depends on the number of times the method calls itself. It is essential to point out that because the algorithms are executed before the schedules are sent to the MAS, the time spent on readjustment does not influence the performance of the FMS.

IX. CONCLUSION

This paper presented a multi-agent architecture for flexible manufacturing systems (FMS), emphasizing the applicability of consensus in manufacturing control problems. In the literature, consensus algorithms have rarely been adapted to deal with production control or have not yet been implemented in flexible manufacturing systems with the necessary depth and detail to enable portability to other production environments, enhancements, and incorporation into commercial solutions. This paper presented the architecture of the proposed multi-agent system (MAS), consensus model, agent model, format of the production schedules, and production control model. MAS can be considered an enabling and enabling technology for Industry 4.0 owing to its dynamic and distributed characteristics, including the ability to modularize, cooperate, and self-organize to solve manufacturing problems and for more intelligent production.

Related studies and the importance of readjusting the production schedules generated by classical predictive scheduling processes were also discussed. The readjustment three-stage filter was developed and evaluated to readjust the predictive production schedules generated by the flexible job-shop scheduling (FJS) algorithm. The first filter serializes null-buffer events to ensure that there are no deadlocks in the FMS part-movement system. The second filter prioritizes the execution of null-buffered events in parallel with other events and prunes certain buffer-limited events to reduce the makespan. The third filter detects and prevents overlapping production resources owing to circular waiting, which is essential for the MAS to execute deadlock-free schedules. Experiments proved that classic predictive scheduling algorithms combined with MAS and filters for readjusting schedules to avoid deadlocks guarantee flexibility of the FMS for production demands.

To the best of our knowledge, there is no information on flexible manufacturing systems in real environments operating with multiple agents and methods similar to those proposed in this study for comparison purposes. In the literature, some proposals are limited to theoretical results, with simplifications that make it impractical to port solutions to real environments or even reproduce them in other studies. Considering that the buffer capacity of the resources and transportation system used to move the parts or products in the production system is crucial to the applicability of the solution, these issues cannot be simplified in studies. Similar to the simulation platform presented in this study, many real manufacturing environments have limitations in terms of the system for moving parts around a factory. An example is the overhead cranes used by hundreds of factories, where the approach presented in this study can be applied.

The approaches proposed are suitable for manufacturing systems, especially those that require sequential operations. However, the weakness of the approach is that the more the manufacturing machines are shared, higher is the unavailability for parallel operations, which increases the production

time. These situations occur because of serializing particular operations of the same job owing to limited buffer resources and transport restrictions to avoid deadlocks. One solution is to increase the number of machines performing the same type of operation and the buffer capacity, or to limit the number of distinct products in production schedules.

Some issues are challenging to address in the readjustment of production schedules, such as unforeseen runtime situations, production resource failures, and urgent orders. In these cases, production rescheduling may be the focus of research, but preferably independent of the architecture and may be complementary to the approach proposed in this study, and vice versa. In future studies, new filters will be developed to handle the relationship between the trajectory and processing time to reduce the makespan. A future goal is to develop an approach for agents to obtain partial group consensus when rescheduling is required. Industry 4.0 technologies are essential to improve decision-making about readjustment and rescheduling. The main challenge in decision-making processes is the availability of information that supports understanding problems and choosing solutions.

REFERENCES

[1] A. R. Boccella, P. Centobelli, R. Cerchione, T. Murino, and R. Riedel, "Evaluating centralized and heterarchical control of smart manufacturing systems in the era of industry 4.0," *Appl. Sci.*, vol. 10, no. 3, p. 755, Jan. 2020.

[2] M. Javaid, A. Haleem, R. P. Singh, and R. Suman, "Enabling flexible manufacturing system (FMS) through the applications of industry 4.0 technologies," *Internet Things Cyber-Phys. Syst.*, vol. 2, pp. 49–62, Jan. 2022.

[3] D. Mourtzis, J. Angelopoulos, and G. Dimitrakopoulos, "Design and development of a flexible manufacturing cell in the concept of learning factory paradigm for the education of generation 4.0 engineers," *Proc. Manuf.*, vol. 45, pp. 361–366, Jan. 2020.

[4] R. V. Barenji, A. V. Barenji, and M. Hashemipour, "A multi-agent RFID-enabled distributed control system for a flexible manufacturing shop," *Int. J. Adv. Manuf. Technol.*, vol. 71, nos. 9–12, pp. 1773–1791, Apr. 2014.

[5] M. Wooldridge, *An Introduction to MultiAgent Systems*, 2nd ed. Hoboken, NJ, USA: Wiley, 2009.

[6] B. Denkena, M.-A. Dittrich, S. Fohlmeister, D. Kemp, and G. Palmer, "Scalable cooperative mulit-agent-reinforcment-learning for order-controlled on schedule manufacturing in flexible manufacturing systems," in *Simulation in Produktion und Logistik 2021*, J. Franke and P. Schuderer, Eds. Göttingen, Germany: Cuvillier Verlag, 2021, pp. 305–314.

[7] V. C. Kalempa, L. Piardi, M. Limeira, and A. S. De Oliveira, "Fault-resilient collective ternary-hierarchical behavior to smart factories," *IEEE Access*, vol. 8, pp. 176905–176915, 2020.

[8] A. L. De Sousa and A. S. De Oliveira, "Distributed MAS with leaderless consensus to job-shop scheduler in a virtual smart factory with modular conveyors," in *Proc. Latin Amer. Robot. Symp. (LARS), Brazilian Symp. Robot. (SBR) Workshop Robot. Educ. (WRE)*, Nov. 2020, pp. 1–6.

[9] T. T. Mezgebe, H. B. E. Haouzi, G. Demesure, R. Pannequin, and A. Thomas, "Multi-agent systems negotiation to deal with dynamic scheduling in disturbed industrial context," *J. Intell. Manuf.*, vol. 31, no. 6, pp. 1367–1382, Aug. 2020.

[10] G. Weiss, *Multiagent Systems*. Cambridge, MA, USA: MIT Press, 2013.

[11] Y. Li and C. Tan, "A survey of the consensus for multi-agent systems," *Syst. Sci. Control Eng.*, vol. 7, no. 1, pp. 468–482, 2019.

[12] I. R. Uhlmann and E. M. Frazzon, "Production rescheduling review: Opportunities for industrial integration and practical applications," *J. Manuf. Syst.*, vol. 49, pp. 186–193, Oct. 2018.

[13] L. Hu, Z. Liu, W. Hu, Y. Wang, J. Tan, and F. Wu, "Petri-net-based dynamic scheduling of flexible manufacturing system via deep reinforcement learning with graph convolutional network," *J. Manuf. Syst.*, vol. 55, pp. 1–14, Apr. 2020.

[14] M. Zhang, F. Tao, and A. Y. C. Nee, "Digital twin enhanced dynamic job-shop scheduling," *J. Manuf. Syst.*, vol. 58, pp. 146–156, Jan. 2021.

[15] A. Ham, "Transfer-robot task scheduling in job shop," *Intl. J. Prod. Res.*, vol. 10, no. 5, pp. 1–11, 2020.

[16] J.-Q. Li, J.-W. Deng, C.-Y. Li, Y.-Y. Han, J. Tian, B. Zhang, and C.-G. Wang, "An improved Jaya algorithm for solving the flexible job shop scheduling problem with transportation and setup times," *Knowl.-Based Syst.*, vol. 200, Jul. 2020, Art. no. 106032.

[17] J. K. Mogali, L. Barbulescu, and S. F. Smith, "Efficient primal heuristic updates for the blocking job shop problem," *Eur. J. Oper. Res.*, vol. 295, no. 1, pp. 82–101, Nov. 2021.

[18] K. Li, Q. Deng, L. Zhang, Q. Fan, G. Gong, and S. Ding, "An effective MCTS-based algorithm for minimizing makespan in dynamic flexible job shop scheduling problem," *Comput. Ind. Eng.*, vol. 155, May 2021, Art. no. 107211.

[19] Z. Gao, Y. Feng, and K. Xing, "A hybrid estimation-of-distribution algorithm for scheduling flexible job shop with limited buffers based on Petri nets," *IEEE Access*, vol. 8, pp. 165396–165408, 2020.

[20] Y. Sun, S.-H. Chung, X. Wen, and H.-L. Ma, "Novel robotic job-shop scheduling models with deadlock and robot movement considerations," *Transp. Res. E, Logistics Transp. Rev.*, vol. 149, May 2021, Art. no. 102273.

[21] K.-W. Hansmann, M. Hoeck, and K.-W. Hansmann, "Production control of a flexible manufacturing system in a job shop environment," *Int. Trans. Oper. Res.*, vol. 4, nos. 5–6, pp. 341–351, Nov. 1997.

[22] Z. H. Ismail and N. Sariff, "A survey and analysis of cooperative multi-agent robot systems: Challenges and directions," in *Applications of Mobile Robots*, E. G. Hurtado, Ed. Rijeka, Croatia: IntechOpen, 2019, ch. 1, pp. 1–22.

[23] A. Fazlirad and R. W. Brennan, "Multiagent manufacturing scheduling: An updated state of the art review," in *Proc. IEEE 14th Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2018, pp. 722–729.

[24] P. Meilanitasari and S.-J. Shin, "A review of prediction and optimization for sequence-driven scheduling in job shop flexible manufacturing systems," *Processes*, vol. 9, no. 8, p. 1391, Aug. 2021.

[25] J. Luo, M. Zhou, and J. Q. Wang, "AB&B: An anytime branch and bound algorithm for scheduling of deadlock-prone flexible manufacturing systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 18, no. 4, pp. 2011–2021, Oct. 2021.

[26] M. Bi, I. Kovalenko, D. M. Tilbury, and K. Barton, "Dynamic resource allocation using multi-agent control for manufacturing systems," *IFAC-PapersOnLine*, vol. 54, no. 20, pp. 488–494, 2021.

[27] S. Baer, J. Bakakeu, R. Meyes, and T. Meisen, "Multi-agent reinforcement learning for job shop scheduling in flexible manufacturing systems," in *Proc. 2nd Int. Conf. Artif. Intell. Industries (AI4I)*, Sep. 2019, pp. 22–25.

[28] Z. Han, Y. Liu, H. Shi, and X. Tian, "A dynamic buffer reservation method based on Markov chain to solve deadlock problem in scheduling," in *Proc. 11th Int. Conf. Modeling, Identificat. Control*, R. Wang, Z. Chen, W. Zhang, and Q. Zhu, Eds. Singapore: Springer, 2020, pp. 1205–1213.

[29] M. Durasević and D. Jakobović, "Comparison of schedule generation schemes for designing dispatching rules with genetic programming in the unrelated machines environment," *Appl. Soft Comput.*, vol. 96, Nov. 2020, Art. no. 106637.

[30] S. Luo, "Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning," *Appl. Soft Comput.*, vol. 91, Jun. 2020, Art. no. 106208.

[31] P. Kianpour, D. Gupta, K. K. Krishnan, and B. Gopalakrishnan, "Automated job shop scheduling with dynamic processing times and due dates using project management and industry 4.0," *J. Ind. Prod. Eng.*, vol. 10, no. 5, pp. 1–14, 2021.

[32] X. Liang, Y. Huang, and M. Huang, "Prediction of optimal rescheduling mode of flexible job shop under the arrival of a new job," in *Proc. IEEE 8th Int. Conf. Comput. Sci. Netw. Technol. (ICCSNT)*, Nov. 2020, pp. 55–58.

[33] R. H. Caldeira, A. Gnanavelbabu, and T. Vaidyanathan, "An effective backtracking search algorithm for multi-objective flexible job shop scheduling considering new job arrivals and energy consumption," *Comput. Ind. Eng.*, vol. 149, Nov. 2020, Art. no. 106863.

[34] G. Xu and Y. Chen, "Petri-net-based scheduling of flexible manufacturing systems using an estimate function," *Symmetry*, vol. 14, no. 5, p. 1052, May 2022.

[35] A. Teymourifar, G. Ozturk, Z. K. Ozturk, and O. Bahadir, "Extracting new dispatching rules for multi-objective dynamic flexible job shop scheduling with limited buffer spaces," *Cognit. Comput.*, vol. 12, no. 1, pp. 195–205, Jan. 2020.
[36] A. Dabah, A. Bendjoudi, A. AitZai, and N. N. Taboudjemat, "Efficient parallel Tabu search for the blocking job shop scheduling problem," *Soft Comput.*, vol. 23, no. 24, pp. 13283–13295, Dec. 2019.
[37] J. Lange and F. Werner, "On neighborhood structures and repair techniques for blocking job shop scheduling problems," *Algorithms*, vol. 12, no. 11, p. 242, Nov. 2019.
[38] V. D. Sasso, L. Lamorgese, C. Mannino, A. Onofri, and P. Ventura, "The tick formulation for deadlock detection and avoidance in railways traffic control," *J. Rail Transp. Planning Manage.*, vol. 17, Mar. 2021, Art. no. 100239.
[39] M. Bashir, "Optimal enforcement of liveness for decentralized systems of flexible manufacturing systems using Petri nets," *Trans. Inst. Meas. Control*, vol. 42, no. 12, pp. 2206–2220, Aug. 2020.
[40] F. Čapkovič, "Dealing with deadlocks in industrial multi agent systems," *Future Internet*, vol. 15, no. 3, p. 107, Mar. 2023.
[41] N. Du, H. Hu, and M. Zhou, "Robust deadlock avoidance and control of automated manufacturing systems with assembly operations using Petri nets," *IEEE Trans. Autom. Sci. Eng.*, vol. 17, no. 4, pp. 1961–1975, Oct. 2020.
[42] M. Hu, S. Yang, and Y. Chen, "Partial reachability graph analysis of Petri nets for flexible manufacturing systems," *IEEE Access*, vol. 8, pp. 227925–227935, 2020.
[43] H. Kaid, A. Al-Ahmari, Z. Li, and R. Davidrajuh, "Single controller-based colored Petri nets for deadlock control in automated manufacturing systems," *Processes*, vol. 8, no. 1, p. 21, Dec. 2019.
[44] S. Messinis and G. Vosniakos, "An agent-based flexible manufacturing system controller with Petri-net enabled algebraic deadlock avoidance," *Rep. Mech. Eng.*, vol. 1, no. 1, pp. 77–92, Dec. 2020.
[45] H. Z. Nabi and T. Aized, "Performance evaluation of a carousel configured multiple products flexible manufacturing system using Petri net," *Oper. Manage. Res.*, vol. 13, nos. 1–2, pp. 109–129, Jun. 2020.
[46] A. Amirkhani and A. Barshooi, "Consensus in multi-agent systems: A review," *Artif. Intell. Rev.*, vol. 55, pp. 1–39, Nov. 2021.
[47] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-agent systems: A survey," *IEEE Access*, vol. 6, pp. 28573–28593, 2018.
[48] (2023). *ROS-Industrial*. [Online]. Available: https://rosindustrial.org/
[49] T. Alves. (2023). *OpenPLC*. [Online]. Available: https://openplcproject.com/
[50] K. Conley, D. Thomas, and J. Perron. (2023). *Python Client Library for ROS*. [Online]. Available: http://wiki.ros.org/rospy

ALEX LUIZ DE SOUSA received the degree in information systems technology and the master's degree in electrical engineering from Santa Catarina State University (UDESC). He is currently pursuing the Ph.D. degree with the Federal Technological University of Paran (UTFPR). He is also a Professor with UDESC. His research interests include intelligent production systems, autonomous mobile robots, distributed systems, multi-agent systems, and industrial networks.

ANDRE SCHNEIDER DE OLIVEIRA (Member, IEEE) received the Graduate degree in computer engineering from the University of Vale do Itaja (Univali), in 2004, and the master's degree in mechanical engineering and the Ph.D. degree in automation and systems engineering from the University Federal of Santa Catarina (UFSC), in 2007 and 2011, respectively. He is currently an Associate Professor with the Federal Technological University of Paran (UTFPR). He is also a permanent Professor with the Graduate Program in Electrical Engineering and Industrial Informatics, where he worked as the Head of the Department (2018–2021). He works in the field of electrical engineering with an emphasis on robotics, mechatronics, and automation, working mainly with the themes, such as navigation and localization; autonomous and intelligent systems, perception and identification of environments; cognition and deliberative decisions; and industrial multi-agent systems.