Festival kvaliteta 2025
ISBN 978-86-6335-122-6

Gligorije Mirkov [1]
Miladin Stefanović

Research paper
DOI – 10.24874/QF.25.077

APPLICATION OF AGENTS IN FMS: AN EDUCATIONAL APPROACH

Abstract: *This paper explores the application of multi-agent systems (MAS) in an educational environment through the modeling and simulation of a flexible manufacturing system (FMS). The focus is on developing models and programming agents for managing CNC machines, a robot with a peripheral axis, and part manipulation. Students are guided through recommended exercises aimed at optimizing operations and handling parts. Using the Python programming language, students develop code that enables both simulation and real-world application in laboratory conditions. Additionally, through RFID part identification and API communication, the system allows for automation and intelligent control of manufacturing processes. This approach fosters the development of Industry 4.0 concepts in education and enhances students' practical skills in the domain of digital manufacturing.*

Keywords: *Multi-agent systems, FMS, CNC, robotics, DNC, OPC UA, RFID, education, Industry 4.0*

1. Introduction

In modern manufacturing systems, the concept of agents is increasingly taking center stage due to its ability to enable flexibility, autonomy, and intelligent management of complex processes. Agents are software entities that act according to defined objectives, capable of independently making decisions and executing actions based on data received from their environment. This approach offers numerous advantages in industrial applications, where complex tasks, variable working conditions, and the need for increased efficiency are frequently encountered.

Flexible manufacturing systems (FMS) are a key component of modern industrial plants, where agents are used to control and coordinate machines, robots, and transport systems, enabling timely adaptation to changes in production. In Industry 4.0, which emphasizes digitalization, interconnectivity, and data utilization, the role of agents becomes even more significant. Intelligent agents leverage artificial intelligence (AI) and machine learning (ML) techniques to process real-time data, make quick decisions, and improve processes in increasingly complex environments.

This paper examines the characteristics of agents and their application in FMS, emphasizing the development of intelligent agents capable of adapting to different conditions and tasks. Special attention is given to machine learning techniques, which enable agents to improve performance through experience, optimize processes, and reduce the need for human intervention.

1.1. Agent-Based Manufacturing Architectures

With the rapid expansion of computing technology, network and communication structures, information exchange, and sensor

[1] Corresponding author: Gligorije Mirkov
Email: gmirkov@sbb.rs

development, conditions have been created for the development of agent-based control architectures. In most studies, including this one, knowledge and experiences related to agents have been utilized, with a particular focus on their attributes, such as autonomy, self-responsibility, and self-recovery, making these technologies fundamental to future manufacturing systems (van Dyke Parunak, 1998).

An agent-based control architecture is characterized by a decentralized system structure with the accelerated growth of computer technology, synchronization, communication, and information exchange, which serve as the foundation for further advancements. The software system, agent-based, is seen as a framework for computation and software development.

(Russell, S., & Norvig, P. , 2010) categorize agents into four types:

- **Simple reflex agents** – select actions based on basic perception without considering past observations.
- **Model-based reflex agents**– incorporate a model of their environment, tracking changes over time.
- **Goal-based agents** – enhance control by introducing goals that define desirable situations.
- **Utility-based agents** – extend goal-based agents by assigning a utility measure to different outcomes.

Since agent-based systems lack a universally accepted definition, there are generally two abstractions (Monostori, L., Váncza, J., & Kumara, S. R. T., 2006):

- An agent is a computing system operating in a dynamic environment, capable of exhibiting autonomous and intelligent behavior.
- An agent may have an environment that includes other agents, forming a multi-agent system (MAS).

According to Monostori, Váncza, & Kumara (2006), agents are characterized by the following key aspects:

- Agents act on behalf of their designer or user to fulfill a specific purpose.
- Agents are autonomous, meaning they control their internal state and behavior.
- Agents demonstrate intelligence, ranging from fixed rule applications to reasoning, planning, and learning.
- Agents can be heterogeneous.
- Agents interact with their environment and with other agents in a community.
- Ideally, agents are adaptive and capable of adjusting their behavior to environmental changes without intervention from their designer.

From these characteristics, it can be concluded that the fundamental property of an agent is its ability to make independent decisions and react to environmental changes. In many models, agents may not possess all these characteristics, as this depends on the nature and complexity of the system being modeled. The rules governing agent behavior can range from simple to highly complex, depending on the volume and type of information available for decision-making.

Agents exhibit built-in behaviors that enable them to make independent decisions; they are not passive but react to inputs from other agents or the environment to achieve predefined objectives. Each agent has clearly defined boundaries, distinguishing its attributes from shared system attributes. Agent behavior can be determined by simple behavioral rules or by highly complex adaptive responses. Over time, agents transition through different states, where the state of an agent at a given moment is defined by its attributes. The state of an agent-based model depends on the individual states of all agents in the system and the state of the environment.

Agents engage in dynamic interactions with other agents, which causally define their behavior. They are characterized by built-in communication protocols for interacting with other agents. Agents can manage resources or acquire resources through interactions with other agents, further distinguishing them from one another. Multi-agent systems create a network of agents that collaborate to achieve shared goals.

The process of building an agent-based model typically follows these fundamental phases:

- **Formulating research questions** – clearly defining the problem and objectives.
- **Hypothesis development** – establishing hypotheses for specific processes and structures.
- **Defining model structure** – creating a document describing the model's framework.
- **Model implementation** – transforming the conceptual model into a mathematical or computational representation.
- **Analysis, testing, and validation** – refining and verifying the model's accuracy.

In addition to these fundamental phases, other critical steps in agent-based modeling include defining agent behavior (from simple to complex), identifying agent behaviors, selecting the development platform and strategy, acquiring necessary data, validating agent behavior, and executing the model.

2. Agent Communication Languages

Agent-based models can be implemented using general-purpose programming languages and tools or within environments specifically designed for this type of modeling. Each of these software modeling approaches has distinct characteristics and methods for addressing problems. One key feature is communication, which requires appropriate languages and communication protocols such as:

- **ACL** (Agent Communication Language)
- **KQML** (Knowledge Query and Manipulation Language)
- **FIPA ACL** (Foundation for Intelligent Physical Agents)

Additionally, general-purpose programming languages that support the development and implementation of agent architectures can also be used. The key characteristics of these languages are outlined in Table 1.

Table 1. Characteristics of Specialized Agent Communication Languages

Specialized Agent Communication Languages		Characteristics
ACL	(Agent Communication Language)	- **General Purpose**: Designed for communication between agents in distributed systems. - **Inter-Agent Communication**: Enables agents to exchange information, make requests, provide offers, or execute actions. - **Standardized Performative Actions**: Uses performatives such as "REQUEST," "INFORM," and "QUERY" to define communication intent. - **Formal Semantics**: Messages in ACL have a well-defined structure with clear semantics, ensuring comprehension of communication. - **Flexibility**: Supports various types of communication, including requests, responses, and negotiations.

KQML	(Knowledge Query and Manipulation Language)	- **Knowledge Exchange**: Designed for knowledge sharing in distributed environments. - **Abstract Communication Layer**: Provides a high-level means of sending requests or information between agents without focusing on their internal implementation. - **Performatives**: Similar to ACL, it uses performatives such as "ASK," "TELL," and "ACHIEVE" to express intentions like querying or issuing commands. - **Modularity**: Supports communication independent of a specific application domain, offering mechanisms for negotiation, task delegation, and knowledge exchange. - **Extensibility**: Allows the integration of specialized protocols for different types of agent interactions.
FIPA ACL	(Foundation for Intelligent Physical Agents - ACL)	- **Standardized Agent Language**: Developed by FIPA as a standard for agent communication. - **Performatives and Dialogues**: Uses performatives such as "INFORM," "REQUEST," and "CONFIRM" to structure agent interactions. - **Well-Defined Semantics**: Based on logical models (Beliefs, Desires, Intentions – BDI). - **Support for Complex Interactions**: Enables agents to conduct multi-step dialogues, including queries, goal setting, and negotiations. - **Compatibility with Different Environments**: Designed to be flexible and adaptable for use in various agent-based systems and distributed environments.

Based on the previous information provided in Table 1, the similarities/differences of these languages are outlined. ACL and KQML share similar characteristics in terms of performatives and enabling communication between agents, while FIPA ACL is a more advanced and standardized language with more precisely defined rules for agent communication, emphasizing interoperability and application in complex systems.

Comparing the languages ACL, KQML, and FIPA ACL with the Python programming language, as presented in Table 2, requires focusing on specific aspects, as Python is not a direct agent communication language but rather a general-purpose programming language.

Table 2. Comparison of Specialized Languages with a General-Purpose Language (Python)

	ACL, KQML, FIPA ACL	Python
Purpose	Specialized for inter-agent communication in multi-agent systems (MAS). Primarily used for exchanging information, making requests, and negotiating using performatives like "REQUEST" and "INFORM".	A general-purpose programming language used for a wide range of applications, including MAS development, backend programming, and data science.
Language Structure	These languages are highly abstract and structural, focusing on performatives and message semantics. Their sentences are short and have clearly defined components (sender, receiver, content, performative). These languages themselves do not define the logic of the agent's behavior, but rather the method of communication. Example of a KQML message: (ask :sender AgentA :receiver AgentB :content (temperature ?x))	Python is an imperative language used to express the behavior of agents, algorithms, and other functionalities. In Python, agents and their logic can be implemented, while communication messages such as ACL and KQML are just a part of the functionality. -Python implements message sending using libraries such as pika for RabbitMQ[1] or libraries for multi-agent systems. -Python code for an agent might look like this: def send_message(agent, message): agent.receive(message)

[1] RabbitMQ is a software system for message exchange between different applications or system components, allowing applications to communicate with each other through message exchange without the need for a direct connection.

	ACL, KQML, FIPA ACL	Python
Communication	Specifically developed for decentralized communication between agents.	Python does not have a built-in standard for agent communication but can implement communication protocols such as HTTP, TCP/IP, or specialized MAS libraries like Spade and Jade.
Role in Multi-Agent Systems	These languages represent the core of communication between agents. The focus is on enabling flexible and understandable communication. They define how agents communicate, exchange information, and negotiate.	Python is a very flexible language that can be used to build a complete multi-agent system. While ACL, KQML, and FIPA ACL function as communication languages, Python can implement all aspects of agents, including logic, behavior, and communication between agents, using these languages or its own protocols.
Flexibility and Extensibility	These languages are more narrowly specialized and are used for specific interactions. While they are flexible within their domain, they do not support other programming paradigms, such as object-oriented programming, procedural, or functional programming.	Python is a general-purpose programming language that supports a wide range of programming paradigms (procedural, object-oriented, functional). It can be used for all stages of agent development – from behavior logic to communication and learning.
Libraries and Ecosystem	These languages do not have direct ecosystem support like Python. There are implementations that enable their use in specific systems.	Python has a rich ecosystem with numerous libraries, including tools for artificial intelligence, machine learning, multi-agent systems (e.g., Spade, Jade), as well as support for protocols that enable agent communication.

Based on the previous discussion, ACL, KQML, and FIPA ACL are specialized languages for agent communication in multi-agent systems, while Python is a multi-purpose programming language that enables the implementation of agents, their logic, and communication protocols, with the ability to use these languages or, on the other hand, develop its own communication tools.

The choice of the appropriate programming language is crucial for easier mastery of the basics and further progress. Learning about agents is a complex field that relies on the principles of artificial intelligence, distributed systems, and automation. Therefore, it is important to choose a language that strikes a balance between simplicity and power, so that beginners can effectively learn and apply concepts in real-world scenarios.

The programming language should be accessible, with extensive documentation and support for agents, but also flexible enough to implement different types of agents, such as autonomous, reactive, or intelligent agents. Furthermore, it is important that the chosen language is relevant for the industry and research projects in the field of artificial intelligence and Industry 4.0, so that the tools learned can be easily applied in a professional environment.

For these reasons, Python emerges as a logical choice for the first language in agent programming. The key features of Python are outlined in the previously mentioned Table 3. Due to all these factors, Python is an excellent choice for beginners in agent programming, as it allows for a gradual transition from simple tasks to more complex challenges.

2.1. Studying and Practicing Agent Programming

Studying and practicing agent programming for future programmers is becoming increasingly significant, especially in the context of Industry 4.0, which relies on the digitalization and automation of production processes. Multi-agent systems play a key role in this context because they enable decentralized management, autonomous decision-making, and adaptive behavior in complex environments. These systems consist of agents that can be physical entities, such as machines and robots, or software

entities that communicate, negotiate, and coordinate activities to achieve set goals.

In Industry 4.0, smart factories and flexible manufacturing systems require dynamic adaptability, and agents can enhance system performance through autonomous management, self-organization, and real-time data-driven learning. In practice, agents can manage the operation of CNC machines, industrial robots, logistics systems, and communication between these components. Programming agents requires an understanding of distributed systems, artificial intelligence (AI), and communication protocols like ACL, KQML, and FIPA-ACL, which are skills that are essential for working in modern industrial environments.

Practicing agent programming allows students to develop the necessary competencies to design and implement such systems, including the ability to design autonomous systems, integrate artificial intelligence, and optimize the operation of industrial systems. Additionally, practicing agent programming provides practical knowledge of tools such as Python, RabbitMQ, and other software tools, making those trained in this field more competitive in the labor market, which is increasingly oriented towards digital transformation and the implementation of Industry 4.0 concepts.

Agent-based architectures have been developed for various types and purposes in manufacturing

Table 3. Key Characteristics of Python Language

Python Characteristics	Explanation
Simplicity and Readability	Python is known for its simple syntax, resembling natural language, making it ideal for beginners. This facilitates a quick understanding of basic programming concepts, including agent implementation.
Library Support	Python has a highly developed standard library and numerous additional libraries that support agent development. For example, libraries such as spaCy for natural language processing, TensorFlow and scikit-learn for machine learning, and Pykka for implementing the actor model make it easier to write complex agent-based systems.
Support for Agents	Python supports procedural, object-oriented, and functional programming, allowing users to explore different programming paradigms, which is beneficial for agent development.
User Community	Python has a vast community of users and developers, providing abundant resources, tutorials, forums, and libraries. Beginners can easily find answers and guidelines for agent development.
Integration with Distributed Systems	Python offers tools for easily setting up distributed systems, which is useful when working with agents in environments such as FMS (Flexible Manufacturing Systems). For instance, Python integrates seamlessly with RabbitMQ, enabling efficient message queue management.
Industry Adoption	Python is widely used in Industry 4.0 for applications related to automation, robotics, machine learning, and the Internet of Things (IoT). Python skills can be easily transferred to Industry 4.0 applications, facilitating the learning and implementation of agents in real industrial systems.

Agents in FMS represent a key technology for managing and optimizing manufacturing processes. Agents are autonomous software entities that operate in dynamic environments and exhibit intelligent behavior, including decision-making, planning, and learning.

The benefits of using agents include increased flexibility, autonomy, and adaptability to environmental changes. These

systems enable decentralized management, reducing the need for a centralized control system and allowing for faster and more efficient responses to changes in the manufacturing process.

2.2. Future Vision of Agent-Based Manufacturing in FMS

The future of flexible manufacturing systems (FMS) based on agents promises numerous advantages and innovations in the industry. Here are some key points that could shape the future of agent-based FMS:

Increased Flexibility and Adaptability

Agents can quickly respond to changes in production requirements, adapting to different production orders and machine configurations, providing dynamic adaptability. They can make real-time decisions based on current conditions, reducing the need for human intervention and autonomously managing processes.

Increased Autonomy

Agents can take over more tasks that previously required human intervention, such as production planning, quality monitoring, and maintenance. Machines equipped with agents can detect and resolve problems without external service intervention.

Integration with the Internet of Things (IoT)

Integrating agents with IoT sensors enables continuous data collection and process optimization based on real-world conditions in the production environment. Real-time data analysis allows for failure prediction and preventive machine maintenance.

Advancement of Collaborative Robots (Cobots)

Agents enable collaborative robots to better understand and interact with human workers, improving safety and efficiency in shared workspaces. Cobots will communicate and share information with other agents and machines, leading to better coordination and synergy.

Artificial Intelligence and Machine Learning

Agents will leverage AI and machine learning to continuously improve performance, analyzing historical data and optimizing processes. By analyzing market trends, agents will be able to predict demand and adjust production accordingly.

Decentralized Control

Agent-based FMS allows for decentralized control, where each agent manages a specific aspect of the system, reducing the risk of system failures due to a central control malfunction. This decentralized structure makes the system more resilient to changes and disruptions, as agents can take over the functions of other agents when necessary.

Resource Optimization

Agents can optimize the use of resources such as energy, materials, and time, reducing costs and increasing efficiency. Precision control of production processes minimizes waste, contributing to sustainability and environmental protection. The future of agent-based FMS promises significant improvements in efficiency, flexibility, and autonomy in manufacturing systems. These technologies enable companies to respond faster to market changes, reduce costs, and enhance product quality, gaining a competitive advantage in the global market.

Agents in flexible manufacturing cells (FMC) play a crucial role in managing and optimizing production processes in the context of Industry 4.0. Industry 4.0, as a smart factory concept, integrates advanced technologies such as the Internet of Things (IoT), artificial intelligence (AI), and big data with traditional manufacturing processes.
This advancement is reflected in:

- Autonomy and decentralization
- Flexibility and adaptability
- Optimization and efficiency
- Predictive maintenance
- Coordination and collaboration

- Improved decision-making

Through these capabilities, agents in FMC play a key role in realizing the vision of Industry 4.0, enabling smart, adaptive, and highly efficient manufacturing systems that can meet the challenges and demands of the modern market.

2.3. The Role of Agents in FMS

Agents in flexible manufacturing systems (FMS) play a crucial role in the Industry 4.0 management model, which focuses on creating smart factories through the integration of digital and physical systems. Industry 4.0 introduces advanced technologies such as IoT, AI, and big data analytics into traditional manufacturing processes. Here's how agents impact these management aspects:

1. Interaction and IoT: Agents enable continuous communication between various machines, sensors, and devices within the manufacturing system. Using IoT, agents collect and share real-time data, ensuring transparency and efficiency in resource management. Agents monitor operations in real-time, quickly detecting and resolving issues, reducing unplanned downtimes, and increasing overall productivity.

2. Artificial Intelligence and Analytics: Agents utilize AI algorithms to analyze sensor data, predicting failures before they occur (predictive maintenance). AI-driven data analytics help optimize production processes by identifying bottlenecks, optimizing workflows, and minimizing waste.

3. Decentralized Decision-Making: Each agent makes independent decisions based on local data and global objectives, reducing the need for centralized control. A decentralized structure allows the system to scale efficiently by adding new agents and resources without major changes to the central management system.

4. Collaboration and Interaction: Multi-Agent Systems (MAS) collaborate to coordinate complex tasks (Wooldridge, 2009). For instance, a transportation agent can coordinate with a production agent to ensure timely material delivery, reducing delays and improving efficiency. Resource management agents optimize the allocation of machines, personnel, and materials according to current production needs (Jennings & Wooldridge, 1998).

5. Flexibility and Adaptability: Agents can dynamically adjust production processes in real-time, responding to shifts in demand or resource availability.

Machine learning techniques enable agents to continuously improve their performance, adapting to new conditions and datasets.

6. Interoperability and Standardization:

Agents use standardized communication protocols, facilitating seamless integration of different systems and technologies within a smart factory. Integration with ERP (Enterprise Resource Planning) systems enables synchronization between manufacturing processes and business operations (Vazquez & Cecilio, 2013).

Agents in FMC significantly enhance production management within Industry 4.0. Through autonomy, flexibility, analytics, and collaboration, they enable intelligent and efficient manufacturing systems capable of adapting to rapidly changing market conditions and demands. This results in increased productivity, reduced costs, and improved product quality (Paulo et al., 2015).

2.4. Agents and Machine Learning Capabilities

Agents in manufacturing systems can possess learning capabilities, particularly through machine learning (ML). In intelligent agent systems, ML enables agents to analyze environmental data, adapt their behavior, and optimize decisions based on experience. This learning capability is crucial for the

development of advanced autonomous systems, such as those used in Industry 4.0, FMS, robotics, and other complex industrial systems. One of the fundamental questions that arise when studying agents is how they learn, in what way, and through which learning stages they progress:

1. Data Collection: Agents gather data from the environment using sensors, user interactions, or other sources.
2. Analysis and Learning: Using various ML algorithms (supervised learning, unsupervised learning, deep learning, etc.), agents analyze data, identify patterns, and derive insights for performance improvement.
3. Behavior Adaptation: Based on analysis, agents adjust future actions for better outcomes.
4. Continuous Improvement: As agents interact more with their environment, their learning ability enhances, leading to more precise and efficient operations.

Examples of agent applications and their impact on the process itself can be observed through:

- Process Optimization: Agents in FMC can learn how to optimize task allocation among CNC machines and robots.
- Failure Prediction: Learning agents can predict potential failures, enabling proactive maintenance and minimizing downtime.

Table 4. Classification of Agents in FMS

Division Agents in FMS		
According to the function	**Production agents**	- Agent machines: Manage the operation of specific machines (eg CNC machines, 3D printers). - Robot Agent: They control the work of industrial robots that perform tasks such as material handling, welding, assembly
	Transport agents	- Internal logistics agent: They manage the transportation of materials and products within the factory (eg autonomous vehicles, conveyor belts).
	Planning agents	- Production planning agent: They plan the production schedule, including the sequence of operations and the allocation of resources. - Maintenance planning agent: They plan and coordinate preventive and corrective maintenance of machines and equipment.
	Supervisory agents	- Quality control agent: They monitor product quality and identify defects. - Safety Agent: They monitor the safety aspects of the production process and ensure compliance with safety standards.
According to the level of control	**Operational agents**	- Act at a lower level of control and manage specific tasks and operations (eg control of machines, execution of production steps).
	Tactical agents	- Make decisions at the intermediate level of control, such as optimization of production lines, scheduling of workers, and coordination between different operating agents.
	Strategic agents	- Operate at a high level of control, managing long-term planning, optimization of resources at the plant-wide level, and making strategic decisions.

According to the way of interaction	**Reactive agents**	- They react to events in real time without prior planning. They are suitable for solving problems quickly and adapting to changes.
	Proactive agents	- They plan their actions in advance based on the prediction of future events and goals. They are focused on optimization and efficiency.
	Hybrid agents	- They combine reactive and proactive characteristics, adapting to the situation and combining quick response and long-term planning.
According to the level of intelligence	**Simple agents**	- Perform basic tasks with minimal intelligence and interaction. They are suitable for routine and repetitive tasks.
	Intelligent agents	- They use advanced algorithms for decision making, learning and optimization. They can use artificial intelligence and machine learning to adapt and improve performance.
According to application domains	**Specialized agents**	- They are focused on specific tasks or processes in production (eg welding agents, assembly agents).
	Generalized agents	- They can be used for different tasks and processes, adapting to different requirements and contexts.
According to the interaction with the environment	**Physical agents**	- Directly manage physical devices and machines in a production environment.
	Virtual agents	Manage information, data, and digital processes, such as production planning, data analysis, and resource optimization.

Machine, enabling proactive maintenance.

- Improving product quality**:** Through the analysis of output data, agents can learn how to enhance the quality of processing or assembly.

This flexibility and learning capability make agents particularly suitable for complex and dynamic environments, such as industrial systems. Machine Learning (ML) within FMS can significantly improve system efficiency and performance. Here are some examples of how ML is used in an FMS environment:

Production Scheduling Optimization

FMS agents utilize machine learning to optimize production schedules based on variables such as machine availability, processing time, or task priorities. Algorithms like genetic algorithms or reinforcement learning can be used to help agents learn how to efficiently allocate tasks to CNC machines or robots.

- Example: If a scheduling agent notices that one machine is frequently overloaded while another remains underutilized, it can adjust the schedule to distribute processing loads more evenly.

Predictive Maintenance of Machines

In FMS, ML models such as neural networks or unsupervised learning can analyze sensor data from CNC machines and robots to predict failures before they occur. These systems collect data such as vibrations, temperature, or machine operating hours, and agents learn patterns that indicate potential failures.

- Example: Based on changes in the vibration frequency of a CNC machine, an agent can predict the need for maintenance or replacement of a specific component or cutting tool.

Product Quality Control

Agents use ML to analyze product quality data during or after processing. Classification algorithms can be applied to detect defects in parts during manufacturing. Cameras and sensors collect data, and agents learn to recognize deviations from standard quality.
- Example: Deep learning algorithms can analyze images of products during CNC machining, identifying micro-defects or surface irregularities that would otherwise go unnoticed.

Energy Resource Optimization

Machine learning can optimize energy consumption in FMS cells by predicting the required energy based on machine load and processing time. Supervised learning can be used to optimize energy resources, reducing the system's energy footprint.
- Example: Agents learn when it is optimal to reduce speed or put machines into standby mode when tasks are less demanding.

Intelligent Robot Navigation

Robots in FMS use ML algorithms for autonomous navigation and path optimization. Reinforcement learning algorithms allow robots to learn the best routes for transporting parts between CNC machines and storage areas, minimizing unnecessary time and energy losses.

Example: If a robotic arm moves between machines and storage, agents can adjust the path in real-time based on congestion or other obstacles.

Adaptive Machine Programming

ML agents learn how to adapt CNC code (e.g., G-code) in real time to optimize machining processes for different materials or complex geometries, enabling more flexible and autonomous manufacturing.

- Example: When machining hard materials, an agent can adjust spindle speed or feed rate based on real-time feedback, optimizing machining quality and reducing tool wear.

Machine learning provides FMS systems with a high degree of flexibility, increases productivity, and enables adaptation to dynamic changes in industrial environments.

2.5. Student Exercises

Students can practice writing different types of agents in Python, as it is a flexible language with a rich set of libraries and tools for working with agents. Table 5. provides examples of several types of agents that students could implement in Python. Similarly, Table 6. presents more complex types of agents for exercises, such as MAS, negotiation agents, and planning agents. The methodological approach to modeling and programming agents is given in Table 7.

Table 5. Types of Agents Students Can Implement in Python

Types of Agents Students Can Implement in Python			
Agent Type	Description	Example	Libraries
Simple Reflex Agents	These agents act based on current perception without considering history.	A robot that moves forward until it encounters an obstacle, then turns.	if-else structures, basic control flow in Python.
Model-Based Reflex Agents	These agents use an internal model of the world to make decisions based on both current and past perceptions.	A robot navigating a maze using previous positions to avoid dead ends.	collections for data structures like deque or set for state tracking.
Goal-Based Agents	These agents have defined goals they try to achieve.	A robot searching for an exit from a maze using search algorithms like A* or Dijkstra.	heapq for priority queues, networkx for graph-based algorithms.

Utility-Based Agents	These agents make decisions based on the utility of different actions.	A robot selecting the most optimal path to a goal while considering factors like energy consumption, time, and safety.	numpy for numerical operations and utility calculations.
Learning Agents	These agents improve their performance through experience.	A robot using Q-learning or deep learning for navigation in a complex environment.	scikit-learn, tensorflow, pytorch for implementing ML algorithms.

Table 6. Složenije vrste agenata za realizaciju vežbi u Pythonu

Challenging Agent Implementations				
Agent Type	Description	Challenges	Example	Libraries
Multi-Agent Systems (MAS)	Systems involving multiple interacting and coordinating agents.	Synchronizing agents, agent communication, conflict avoidance.	A traffic control system where multiple agents (vehicles) coordinate movements to prevent collisions.	mesa for multi-agent modeling, multiprocessing for parallel execution.
Negotiation Agents	Agents that negotiate with each other to achieve common goals.	Developing negotiation strategies, implementing communication protocols.	Agents negotiating resource allocation in a production system.	pydispatch for inter-agent communication.
Planning Agents	Agents that use complex algorithms to plan sequences of actions leading to goal achievement.	Implementing planning algorithms, optimizing plans.	A robot planning a sequence of actions to assemble a product.	pyeda for working with logical formulas, pddlpy for planning domain modeling.

Table 7. Methodological Approach to Exercises in Modeling Different Agent Structures

Agent Type	Task
Simple Reflex Agents	Start with writing simple reflex agents to understand basic principles of state and action management.
Model-Based Reflex Agents	Extend the agent to use an internal world model, which is a natural step toward more complex agents.
Goal-Based Agents	Introduce goals and search algorithms so agents can plan their actions toward defined objectives.
Utility-Based Agents	Add the concept of utility to enable agents to make decisions that optimize specific criteria.
Learning Agents (Machine Learning)	Experiment with basic machine learning techniques to enable agents to learn from experience.
Multi-Agent Systems	Finally, transition to complex systems involving multiple agents that cooperate and communicate with each other.

3. Example of a Reflex Agent in FMC/FMS

3.1. Proposal for Getting Started

For a student who wants to start practicing, the approach given in Table 5. is recommended.

For each of these phases, there are numerous resources and libraries in Python that can help students gradually develop their skills and understanding of agent-based systems.

Scenario: Managing Transport Between Machines

Let's assume we have an agent whose role is to manage the robotic transport system (e.g., SCORBASE-4u) between CNC machines (e.g., PCTURN55 and PCMILL55). The reflex agent will respond to the machine's state and take actions based on simple rules.

Applied Rules:

- If a machine has completed processing, the part needs to be transported to the next machine.
- If a machine has not finished processing, the agent waits until processing is complete.

Pseudo-Code for the Reflex Agent in Python:

```
class ReflexAgent:
    def __init__(self):
        self.machine_status = {"PCTURN55": "busy", "PCMILL55": "idle"}  # Current status of machines
        self.part_clamped = False  # Track if the part is clamped or not

    def perceive_environment(self):
        # Get the current status of the machines (could be from sensors or API)
        return self.machine_status

    def make_decision(self, status):
        # Simple reflex rules based on the current state of machines
        if status["PCTURN55"] == "done" and self.part_clamped:
            self.release_part("PCTURN55")
            self.transport_part("PCTURN55", "PCMILL55")
            self.clamp_part("PCMILL55")
        elif status["PCMILL55"] == "done" and self.part_clamped:
            self.release_part("PCMILL55")
            self.transport_part("PCMILL55", "PCTURN55")
            self.clamp_part("PCTURN55")
        else:
            print("Both machines are busy. Waiting for a machine to finish.")

    def transport_part(self, from_machine, to_machine):
        # Perform the transport operation (simplified)
        print(f"Transporting part from {from_machine} to {to_machine}.")
        # Update machine statuses after transport
        self.machine_status[from_machine] = "idle"
        self.machine_status[to_machine] = "busy"

    def clamp_part(self, machine):
        # Simulate clamping the part in the machine
        print(f"Clamping part in {machine}.")
        self.part_clamped = True

    def release_part(self, machine):
```

```
        # Simulate releasing the part from the machine
        print(f"Releasing part from {machine}.")
        self.part_clamped = False

# Instantiate and run the agent
agent = ReflexAgent()
current_status = agent.perceive_environment()
agent.make_decision(current_status)
```

This simple reflex agent reacts only to the current state of the machine without considering previous actions.

3.2. Explanation of the Reflex Agent's Operation

The agent first perceives the current state of the machines ("busy" or "idle"), which falls into the domain of perception.
Based on the current state, the agent makes decisions. If a machine has completed its task, the agent transports the part to the next machine, which is considered a decision-making process.
The agent then executes the transport step and updates the status of the machines.

The limitations that an agent designer must consider when designing an agent include the fact that a reflex agent does not remember previous states and does not learn from experience. It reacts solely to the current situation, which can be a limiting factor in more complex scenarios, such as prediction or process optimization based on historical data.

In the given example of a reflex agent for managing transportation between CNC machines, the clamping and releasing of the workpiece are not directly addressed. This aspect is crucial in any manufacturing process and usually requires an additional step to control the clamping systems (vises, jaws, or hydraulic systems) on CNC machines or robots.

To complement this example, we can include functions for releasing and clamping the workpiece during transport. These operations would be part of a broader control system that includes CNC machine and robot management, ensuring that the agent properly secures and releases the workpiece before and after transport.

For these reasons, we add functions for:

- Clamping the workpiece on the CNC machine before machining.
- Releasing the workpiece before robot transport.

This is implemented as part of the program that is inserted into the previously written section of the reflex agent program.

Modified Reflex Agent Code with Clamping and Releasing Functions:

```
def clamp_part(self, machine):
# Simulate clamping the part in the machine
print(f"Clamping part in {machine}.")
self.part_clamped = True

def release_part(self, machine):
# Simulate releasing the part from the machine
print(f"Releasing part from {machine}.")
self.part_clamped = False
```

This extended version of the reflex agent incorporates clamping and releasing operations, ensuring the workpiece is handled properly during transportation between CNC machines.

3.3. Clamping and Releasing Process

The function clamp_part() simulates clamping the workpiece on the CNC machine to enable proper machining.

The function release_part() simulates releasing the workpiece from the machine before the robot picks it up and transports it to another machine.

When the workpiece is moved to a new machine (e.g., PCMILL55), the agent simulates clamping the workpiece before starting any machining operation. After the machining is completed, the agent simulates releasing the workpiece from the current machine (e.g., PCTURN55) before the robot can pick it up and transport it further.

This reflex agent manages the robot that moves parts between CNC machines based on the current machine status. It represents a basic example of reactive behavior in FMC/FMS systems. The extended version of this example adds control over clamping and releasing the workpiece, giving the agent greater flexibility and more realistic production process management. These operations are essential to ensure that the robot can safely and efficiently transport the workpiece between CNC machines.

4. Conclusion

The development and implementation of agents in industrial environments, such as Flexible Manufacturing Systems (FMC/FMS) and Industry 4.0**,** represents a key technology for achieving intelligent, adaptive, and automated systems**.** Agents, with their ability to autonomously make decisions and coordinate between different devices**,** significantly contribute to increased efficiency, flexibility, and productivity in manufacturing**.** Establishing communication and synergy between CNC machines and robotic systems through agents enables faster adaptation to changes in production processes**,** reduces downtime**,** and improves resource management**.**

The methodology of agent programming through step-by-step exercises **serves** as a fundamental approach in educating future professionals. The training process can begin with simple reflex agent examples**,** where agents react to specific conditions without memory or complex logic**,** such as making decisions based on machine status**.** Then, students progress to programming agents with more complex behaviors**,** such as goal-oriented agents with basic adaptation capabilities. The next stage introduces communication protocols, allowing agents to exchange information**,** which becomes the foundation for multi-agent systems with collaborative functions**.**

This gradual approach provides students with the opportunity to systematically learn principles and tools needed for agent development**,** enabling them to acquire essential competencies for complex industrial applications**.** This supports a comprehensive understanding and practical skills necessary for Industry 4.0 environments.

Reference:

Jennings, N., & Wooldridge, M. (1998). Applications of Intelligent Agents. In N. M. Jennings, *Agent Technology.* Berlin, Heidelberg: Springer. doi:doi.org/10.1007/978-3-662-03678-5_1

Monostori, L., Váncza, J., & Kumara, S. R. T. (2006). Agent-based systems for manufacturing. . *CIRP Annals*, 55(2), 697-720.

Paulo, L., Armando, W. C., & Stamatis, K. (2015, September). *Industrial Automation based on Cyber-Physical Systems Technologies: Prototype Implementations and Challenges*. doi:10.1016/j.compind.2015.08.004

Russell, S., & Norvig, P. . (2010). Artificial Intelligence: A Modern Approach (3rd ed.). *Prentice Hall*.

van Dyke Parunak, H. (1998). Industrial and practical applications of DAI. In Multiagent Systems. *MIT Press.*, 377-421.

Vazquez, F., & Cecilio, J. . (2013). Integration of multi-agent systems and manufacturing execution systems for decision making in production planning. . *International Journal of Computer Integrated Manufacturing*, 26(10), 890-902.

Wooldridge, M. (2009). An Introduction to MultiAgent Systems, 2nd Edition. Willy. doi:ISBN: 978-0-470-51946-2

Gligorije Mirkov
Belgrade,
Republic of Serbia
gmirkov@sbb.rs
ORCID 0000-0002-1153-0045

Miladin Stefanović
University of Kragujevac, Faculty
of Engineering Sciences
Kragujevac,
Republic of Serbia
miladin@kg.ac.rs
ORCID 0000-0002-2681-0875