applied sciences

MDPI

Article

On the Use of Asset Administration Shell for Modeling and Deploying Production Scheduling Agents within a Multi-Agent System

Vasilis Siatras, Emmanouil Bakopoulos, Panagiotis Mavrothalassitis, Nikolaos Nikolakis and Kosmas Alexopoulos *

Laboratory for Manufacturing Systems and Automation, Department of Mechanical Engineering and Aeronautics, University of Patras, 26504 Patras, Greece; siatras@lms.mech.upatras.gr (V.S.); bakopoulos@lms.mech.upatras.gr (E.B.); mavrothalassitis@lms.mech.upatras.gr (P.M.); nikolakis@lms.mech.upatras.gr (N.N.)
* Correspondence: alexokos@lms.mech.upatras.gr; Tel.: +30-2610-910-160

Abstract: Industry 4.0 (I4.0) aims at achieving the interconnectivity of multiple industrial assets from different hierarchical layers within a manufacturing environment. The Asset Administration Shell (AAS) is a pilar component of I4.0 for the digital representation of assets and can be applied in both physical and digital assets, such as enterprise software, artificial intelligence (AI) agents, and databases. Multi-agent systems (MASs), in particular, are useful in the decentralized optimization of complex problems and applicable in various planning or scheduling scenarios that require the system's ability to adapt to any given problem by using different optimization methods. In order to achieve this, a universal model for the agent's information, communication, and behaviors should be provided in a way that is interoperable with the rest of the I4.0 assets and agents. To address these challenges, this work proposes an AAS-based information model for the description of scheduling agents. It allows multiple AI methods for scheduling, such as heuristics, mathematical programming, and deep reinforcement learning, to be encapsulated within a single agent, making it adjustable to different production scenarios. The software implementation of the proposed architecture aims to provide granularity in the deployment of scheduling agents which utilize the underlying AAS metamodel. The agent was implemented using the SARL agent-oriented programming (AOP) language and deployed in an open-source MAS platform. The system evaluation in a real-life bicycle production scenario indicated the agent's ability to adapt and provide fast and accurate scheduling results.

Keywords: agent-oriented programming; Asset Administration Shell; multi-agent system; scheduling agent

check for updates

Citation: Siatras, V.; Bakopoulos, E.; Mavrothalassitis, P.; Nikolakis, N.; Alexopoulos, K. On the Use of Asset Administration Shell for Modeling and Deploying Production Scheduling Agents within a Multi-Agent System. *Appl. Sci.* **2023**, *13*, 9540. https://doi.org/10.3390/app13179540

Academic Editors: Andrea Prati and Agostino Forestiero

Received: 10 May 2023
Revised: 19 July 2023
Accepted: 16 August 2023
Published: 23 August 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Production scheduling defines when and which of the organization's resources will be used in order to perform a set of operations assigned to the system [1]. Scheduling is a field of production management and is applied in multiple production phases, from the supply chain, production, logistics, and delivery, in order to regulate the performance of an organization. It is a computationally demanding operation that can value the organization's reactivity over the market demand, lower the cost, and achieve faster delivery. Considering the great variety of production and supply chain disturbances, however, scheduling systems need to be adaptable to changes and be able to handle unprecedented scenarios with great accuracy in order to be beneficial for the organization's success.

Within the modern Industry 4.0 (I4.0) schema, which has been the objective for many manufacturing organizations, different manufacturing entities are represented as assets. An asset is the pillar component of I4.0 and symbolizes any entity that has a role within

the I4.0 ecosystem, e.g., machines, production modules, products, software installations, or human resources. The primary goal of I4.0 is to provide interoperability within the system; however, it is due to the Asset Administration Shell (AAS) that this level of interoperability and abstraction can be achieved [2]. AAS is the digital representation of an asset within the I4.0 ecosystem and provides a detailed description of the asset's different domains, e.g., identification, communication, safety, security, etc. The AAS, in essence, is a metamodel, which consists of sub-models (SMs) and sub-model elements (such as collections, properties, events, etc.), which are the basis for describing all the properties of an asset. On top of that, semantics are utilized within each entity to allow higher abstraction as well as accessibility.

Organization management and control systems, which drive the efficiency of industrial processes and achieve automation, are built on top of seamless communication between physical and digital production assets. Agents and Multi-agent systems (MASs) are digital assets with advanced characteristics in autonomous decision making that are deployed within the environment. From the manufacturer's side, interoperability between assets is one of the primary goals, as it is trying to implement solutions that are free from dependencies and have modular characteristics. AAS has been proposed to address this challenge by encapsulating the asset in a digital form that can be accessed from other assets in a standardized way. In the case of software components, this is equally important in order to provide support interoperability with other software and hardware components and build reusable solutions. In the case of agents, it means that information structures, communication pipelines, and operational features will be described in a way that other assets can exploit.

In practice, the AAS concept holds the intermediate layer between the asset and the external world. The reason is that in this way, the asset may not implement custom interaction and communication with other assets but rather use the AAS as the metamodel to describe the domains of the asset and allow third-party components to interact with the asset indirectly through the AAS services and information provided by the hosting platform. This way, even if the asset comes from a different manufacturer or holds different specifications, it will still be able to use the same communication principles. In the case of an agent, however, this may be partially differentiated since communication is also provided within a closed framework, a so-called multi-agent system. In other words, communication among the agents does not need to be hosted by the agent's AAS, as there is already a message-based interaction provided within the framework which hosts these agents. On the other hand, communication with external software and assets is not predefined within the multi-agent system, and the AAS can provide a way for these entities to interact. On top of that, the AAS allows the description of the agent specifications and capabilities, which are provided in order to fit seamlessly within a complex system architecture.

An agent, in general, is an autonomous entity with different capabilities for reacting to external events. The motivation behind multi-agent systems is to achieve decentralized optimization and control of a complex system. As a result, the introduction of the multi-agent system in practice includes a set of different decision-making software that is programmed to react to specific system events. As the system operates, an initial trigger/event may fire the initialization of an agent, which itself may produce one or more events of its own, triggering other agents. This chain reaction of events allows the multi-agent system to produce individual solutions as a subset of a more complex problem. This theoretical implementation of a multi-agent system allows the optimization software to deal with the problem not as a unique optimization block but rather as multiple parts in the pursuit of the overall solution. Nevertheless, the design of the agents, as well as the capabilities that they will provide, is a challenging process. It is necessary that, based on these aspects, the agents will be able to negotiate with each other and solve the problems which they are provided with.

Production scheduling and control is one of the many fields where agents and multi-agent systems can be found beneficial. Scheduling or planning problems are usually complex computational problems that require sophisticated algorithms in order to achieve

the cross-section of accuracy and rapidness. There is, in fact, a large cluster of different scheduling problems, some of which have already found polynomial-reduced algorithms for solving the problems, while others still remain on the side of NP-hard or NP-complete problems. What may support this aspect, however, is the selection of the correct algorithm or method in order to solve each problem based on the nature of the problem itself. This is what may be referred to as meta-scheduling, indicating the ability of the system to exploit different scheduling methods based on the context of the problem. The case, however, is how the software can classify the different problems and provide a solution using the right algorithms. This question could be alternatively stated as how, within a multi-agent system, where different optimization agents coexist, the scheduling agent capabilities can be expressed in a way that will be accessible/interoperable with other assets or agents and have the intelligence to select the highest probable algorithm for any given problem. As such, scheduling operations may be handled in a more cognitive way, allowing the underlying software to make accurate selections when it comes to optimization while at the same time being able to coexist within the Industry 4.0 paradigm supported by the AAS.

This research paper focuses on the provision of an AAS-based information model for the description of scheduling agents in a way that will increase interoperability and allow more efficient decision making. Focusing on the concept of meta-scheduling, i.e., the ability to exploit different algorithms for the solution of a scheduling problem, this paper extends the current state of the art with a generic description model, able to define the agent's capabilities and different algorithms acquired.

This research study aims to support this challenge with the provision of an AAS-based information model for the description of scheduling agents. The proposed modeling methodology describes aspects related to interoperability with other agents or assets and enables abstraction on scheduling operations. The agent software that was created to be aligned with the model is able to encapsulate multiple optimization methods within a toolbox of schedulers, enabling applicability in various industrial cases and scalability. Each of the scheduling methods is parameterized within an AAS model in a way that allows the selection of the corresponding scheduler to be based on the capabilities of the scheduling method. Communication is also established within the AAS based on different operations and events that other agents or assets can access. This enables more abstract interaction across industrial software entities by calling higher-level behaviors of the agent, regardless of the lower-level implementation details. The agent implementation was performed in the SARL agent-oriented programming (AOP) language and deployed in an open-source platform for MASs. During the validation of the scheduling agent, three optimization methods, namely deep reinforcement learning (DRL), mathematical programming, and heuristics optimization, were integrated. The industrial installation took place on a bicycle production system where the scheduling agent was accessed via a shopfloor UI to perform scheduling operations in different problems of the factory.

The rest of the paper is organized as follows: Section 2 presents the state-of-the-art related works of the method that was used in this scientific work. Section 3 presents the proposed AAS model for modeling a Production Scheduling Agent. Moreover, Section 4 presents the industrial implementation in which the proposed model was used. Finally, Section 5 is the conclusion of this work and the future directions of this study.

2. Related Works

Multi-agent systems (MASs) emerged during the 90s as a specialization of Object-Oriented Programming (OOP) with a greater level of responsiveness and autonomous reactions [3]. Since that point, there has been a great amount of standardization involved in MAS development, as well as various technologies that allowed higher-level agent-oriented programming (AOP). According to the definition of O'Hare and Jennings, a network of problem solvers is called an MAS when the solvers work together in order to solve problems that are beyond their individual capabilities [4].

In [5], the authors listed the challenging problems that researchers are facing. The most challenging aspect of designing and developing an MAS is to allow clear intercommunication pathways between the different agents in a way where decisions can be taken through negotiation [6]. This requires agents' awareness of their environment so that a message-based interaction can take place while solving a complex problem [7]. According to [8–13], the MAS design methodologies have three phases: (i) conceptualization, (ii) analysis, and (iii) design [14]. In [14], they presented the stages of the design phase. In [15], authors developed a deterministic coordinated multi-robot exploitation framework in an unknown environment. Moreover, in [16], authors proposed a peer-to-peer environment using generic programming.

Different platforms like JADE [17–20], ZEUS [21–23], Malaca [24], Mad-Kit [25–27], Janus [28,29], NetLogo [30,31], and many more [32–36] have been used in the literature in order to implement MAS. Table 1 organizes the different platforms and references. The agent communication standard is defined by the ACL (Agent Communication Language) standard of the Foundation for Intelligent Physical Agents (FIPA) [32]. In this work, the Janus platform was used. In a further analysis, Janus is an open platform that provides a list of features to develop, run and monitor multi-agent-based applications. In detail, such features are scalability (supports a large number of users), flexibility (provides the developers a range of options to customize their applications), modularity (easy ways to create or remove functionalities), and real-time capabilities (real-time video streaming and surveillance) [28,29,37]. SARL is an agent-oriented programming language that is used for programming in Janus [38,39].

In [40], the preferred applications of agents are discussed, such as planning, scheduling, resource management, strategic decision making, diagnostics, real-time replanning, modeling, ontologies, knowledge integration, software systems, and simulation [41]. Numerous applications of planning and scheduling have been carried out on top of MASs [42,43]. Researchers in the MAS applications in the planning and scheduling area of artificial intelligence have addressed the challenges of enabling agents to act effectively in groups. Machine Learning (ML) approaches allow efficient scheduling of complex production systems by analyzing gathered data in order to enhance overall productivity. Due to the fact that ML algorithms can learn from historical data to identify patterns, they are used to develop predictive models for scheduling [44,45]. Referring to the scheduling problem, many methods and applications have been developed throughout recent years. Approaches that use dispatching rules, whose learning data are developed by preliminary simulation runs, analyze the influence of different parameter values of the selected rule on the system performance [11]. With this tactic, the average amount of tardiness at work may be significantly reduced. Two fields of machine learning general concepts are getting more attention year by year: deep learning (DL) and Reinforcement Learning (RL).

The concept of AAS has been used in many MASs in order to standardize the interaction of the agents [46–50]. Moreover, AASs are a standardized interface for external communication [40,51]. AASs were first presented in [52] by Adolphs et al. (2015). Many scientific works have used AASs, i.e., the use of AASs alongside Digital Twin (DT) [53,54] and AAS alongside DT and reinforcement learning (RL) [55]. Moreover, AASs have been used to parameterize DT, modeling a co-simulation environment [56]. Also, [57] used AASs for the parameterization of Domain-Specific Language (DSL) services. AASs can be used to parameterize AI agents as well, where hyperparameters or information could be exchanged via the AAS concept.

The AAS framework provides a structure for modeling assets, but there may be gaps in how different scheduling agents interpret and implement it, leading to inconsistencies in scheduling data representation and sharing. The AAS framework is designed to capture high-level asset information, but scheduling agents may require more detailed data to manage and optimize production schedules effectively [58]. Integrating scheduling agents with existing AAS-enabled systems can be complex, particularly when different systems use different communication protocols or data formats [54]. Privacy concerns are also

a significant challenge when implementing AAS, particularly when handling sensitive data in production systems, since scheduling data are often sensitive and confidential [59]. Sensitive data can be considered data captures about a physical asset in an industrial environment; such data can include information about performance, operational characteristics, or maintenance. Additionally, scalability is a crucial challenge in the implementation of AAS in large-scale manufacturing systems, which may require further development work to optimize scheduling algorithms and ensure efficient data processing and sharing across different systems [60–62].

AAS models have been used in the literature to capture key information. In [63], the authors design and propose an AAS model for predictive maintenance. Moreover, in [64], the authors provided a comprehensive survey of AAS models for the information and communication layer, where they presented some ontologies used for scheduling, i.e., see [65]. Also, in [66], the authors proposed a modeling approach that was AAS-based using ontologies to overcome the gap of semantic interoperability and allow efficient interactions between Industry 4.0. Comparing the proposed model to the already mentioned works, the contribution of this work is the use of the design of an AAS model for schedulers, where AAS is a standard of Industry 4.0.

Table 1. MAS platforms and frameworks.

Multi-Agent System (MAS) Platforms/Frameworks	Description	Related Works
JADE (Java Agent Development Framework)	JADE is an open-source software framework that is Java-based and used for MASs.	[15–18]
Janus	Janus is an agent-based platform for modeling simulations that enables users to build and test complex systems. It offers a Graphical User Interface (GUI) and a variety of tools.	[26–28]
ZEUS	ZEUS is an open-source agent-based platform implemented in the Java programming language. It provides a GUI and a runtime environment [67].	[19–21]
Mad-Kit (Multi-Agent Development Kit)	Mad-Kit is an agent-based framework for modeling and simulating agents of an MAS. It provides APIs and tools in order to create agents.	[25–27]
Malaca	Malaca is a gent architecture that combines Component-based Software Engineering and Aspect-Oriented Software Development.	[22]
NetLogo	NetLogo is an MAS platform for modeling and simulating agents. It provides a GUI and supports complex systems.	[29,37]
Other platforms	There are other platforms as well, for example, Rapast, AnyLogic, etc.	[30–34]

In the current literature, there is a wide range of approaches where the multi-agent system concept is the primary technology used to solve scheduling or planning problems. On the contrary, there are even more solutions that choose to solve scheduling and planning problems based on monolithic optimization software. A case can be defined for both solutions and the reasoning behind choosing one over the other. What can be highlighted, though, is that complex optimization problems are usually broken down into smaller and simpler problems, which are then handled individually by the system. This is where the multi-agent system could be found beneficial. Being able to deploy different entities and establish the required configuration for these entities to operate and negotiate with each other can provide faster decision making, traceability over the solution, as well as production of different outputs coming from different agents operating in parallel. However, there is a shortage of existing solutions on how these types of agents should be modeled using the AAS concept in order to fit within the I4.0 ecosystem. What is more,

the model should provide granularity in order to exploit different schedule optimization agents that provide different capabilities and methods.

The focus and contribution of this work is to provide an information model using the AAS concept for the description of a scheduling agent and its functionalities. The aim is to bridge the gap on how the interaction between a scheduling agent and other agents or assets can be formalized so that a third-party component can be aware of which services the agent can provide as well as where and how it can access it. This overall goal is to achieve that in an abstract manner, meaning that entities that interact with this agent will not rely on custom interaction methods and will have no dependencies on the implementation technologies of the agent. In detail, the contributions of this work are the following: (1) the use of the AAS concept, where AAS is a standard of Industry 4.0 and (2) the design and discussion of an AAS model to describe scheduling agents, environment, constraints, and objective.

In conclusion, this section presented related works and also mentioned the contributions of this work to the community. Additionally, in the literature, there are several works that use the AAS concept, but they do not provide a generic model for controlling the agents and parameterizing them. The idea of designing the proposed AAS model is to overcome the problem of controlling and parameterizing scheduling agents (please see the Proposed Methodology). Moreover, the design of the AAS models was part of a high-level idea of creating an intelligence (meta-agent) based on the criteria, applicable environments, and objectives of each agent to select the agent that can solve a given scheduling problem.

3. Proposed Methodology

This section introduces an agent architecture (Figure 1) which provides a high level of abstraction to the scheduling problem. The agent shown in Figure 1 could also be referenced as a meta-agent for scheduling, as it aims to use multiple scheduling methodologies while trying to solve the scheduling problem of a production environment. Its key characteristic is that the optimization process is cut off from the core agent logic, which allows the agent to be able to navigate through an arbitrary number of available optimization methods for a given problem. In other words, the agent has access to a toolbox of scheduling solutions, from which it can select based on the type of the problem and the toolbox capabilities. Of course, this raises the issue of accurate scheduler selection for a given problem, which is part of the main logic that is defined within the agent's skills and, in practice, is addressed through the AAS concept. The AAS information model aims to address this issue by allowing a problem–compatibility description for each one of the available scheduling solutions, which can then be used to map the scheduler with requested problems.

The AAS also enables a standardized way of communication between agents as well as other I4.0 systems. In contrast to other assets that use the AAS description to achieve interoperability, it is important to highlight that agents are built on top of the AOP language and message-based interaction to communicate with other agents. As such, on the one hand, agents do not need AAS as the interface for exchanging information within the MAS, but they do need it when it comes to communication with systems that do not belong within the MAS. To this end, the AAS concept serves as the description of communication within the MAS (description of capabilities and messages for agent-to-agent interaction), as well as provides access to entities that do not belong to the MAS.

Each one of the components displayed in Figure 1 is a composite of classes and interfaces. The scheduling services represent the lower level of optimization within the agent, i.e., the scheduling algorithm and mathematical formulation of the problem. This layer consists of an arbitrary number of schedulers, each one aiming to provide a solution to a specific scheduling problem. The algorithm, as well as the type of problems suitable for that scheduler, may differ from one another, thus allowing a wider range of problems suitable for the meta-agent. Each scheduler also expects a different I/O format. For example, in the case that a scheduler is designed to give solutions to the Travel Salesman Problem (TSP), information should be formatted into cities, travel distances, and so on; in

other words, in a format that the scheduler can understand. This brings in the need for the scheduler connector to ensure information is restructured into the format accepted by the scheduler. In addition, it builds a stable connection between the scheduling software and the agent software, which is necessary, especially in cases where the algorithm is executed on a different computational resource.

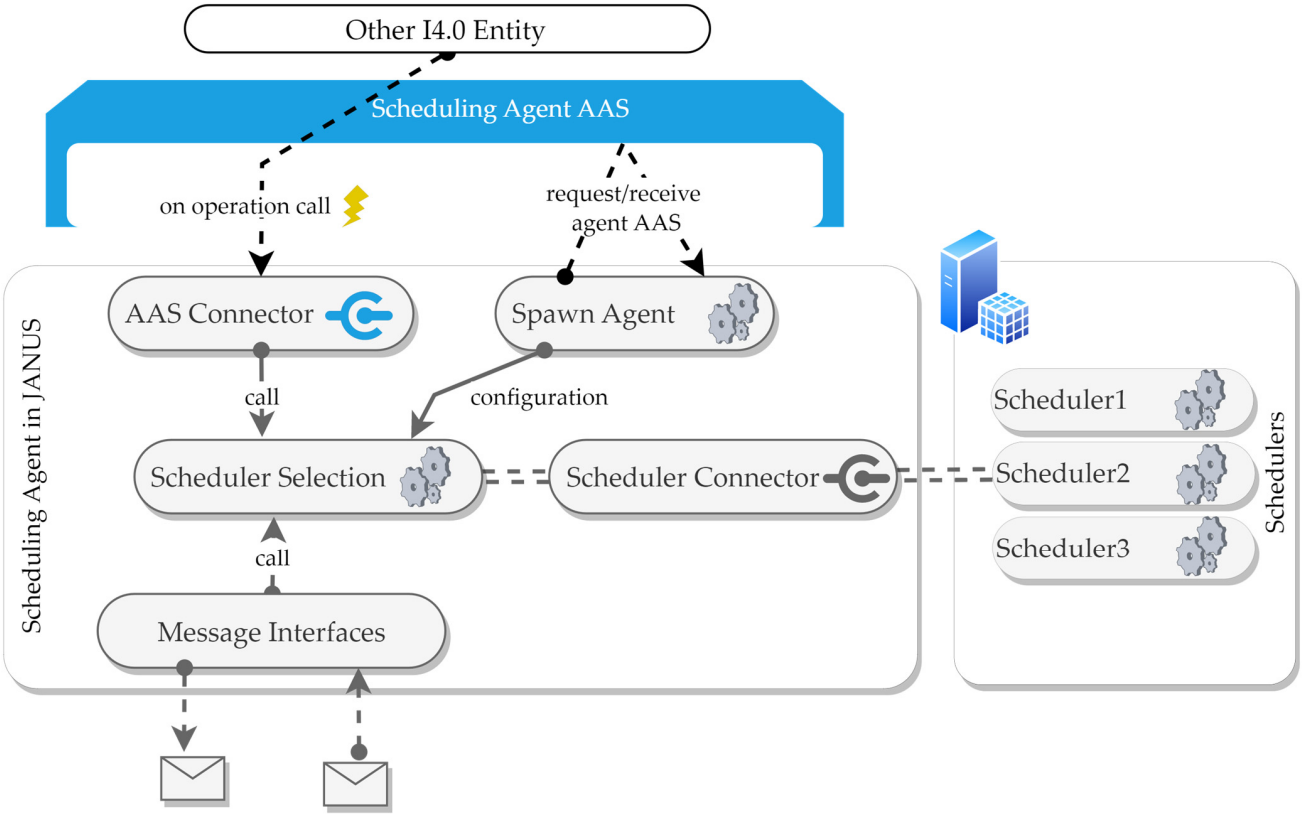


Figure 1. Planning meta-agent software architecture.

One of the primary aspects of this approach is the ability to select among different scheduling methods based on the input problem definition and user requirements. This is what makes the agent a meta-agent and addresses the challenges for abstraction in the scheduling agent. In order for the meta-agent to be able to decide among the different algorithms, it needs to have a classification of the algorithm based on the problems that it is able to solve. This aspect was also supported primarily by the AAS concept, as it allowed the description of the capabilities of each scheduler individually based on the problems that it is able to process. This can set up the compatibility requirements for the scheduler and allow the agent to select among an arbitrary number of schedulers. Within the remainder of the section, the detailed description of the agent AAS is elaborated.

Figure 2 abstractly displays the sub-models (SMs) which compose the AAS of the agent. The AAS always consists of a list of SMs that are clustered within the same AAS; the AAS itself does not have a semantic identifier. However, the SMs do, and thus the SMs could be considered to describe the different domains of the asset. The aim of the proposed list of SMs is to create an abstract description of the agent's functionalities, the way that these functionalities could be accessed by other agents, the information requirements (in other words, the agent's language), and the capabilities in the manner of schedule optimization. Thus, the AAS plays, at first, a configuration role for the agent that is used during the initialization and, secondly, at runtime, a way of interacting with the agent itself using a third-party component. In the second case, as displayed in Figure 2 as well as in Figure 3, the use of a connector is necessary. In the case of AAS, connectors are usually provided by the AAS platform and can be of different kinds (namely, OPC UA, MQTT, and

REST are the most usual ones) and customizable to the asset. In the case of a scheduling agent, an MQTT connector is one of the implementation options which, based on the AAS platform functionalities, acknowledge the agent for any dynamic updates made to the element values of the AAS or the call of a specific operation.

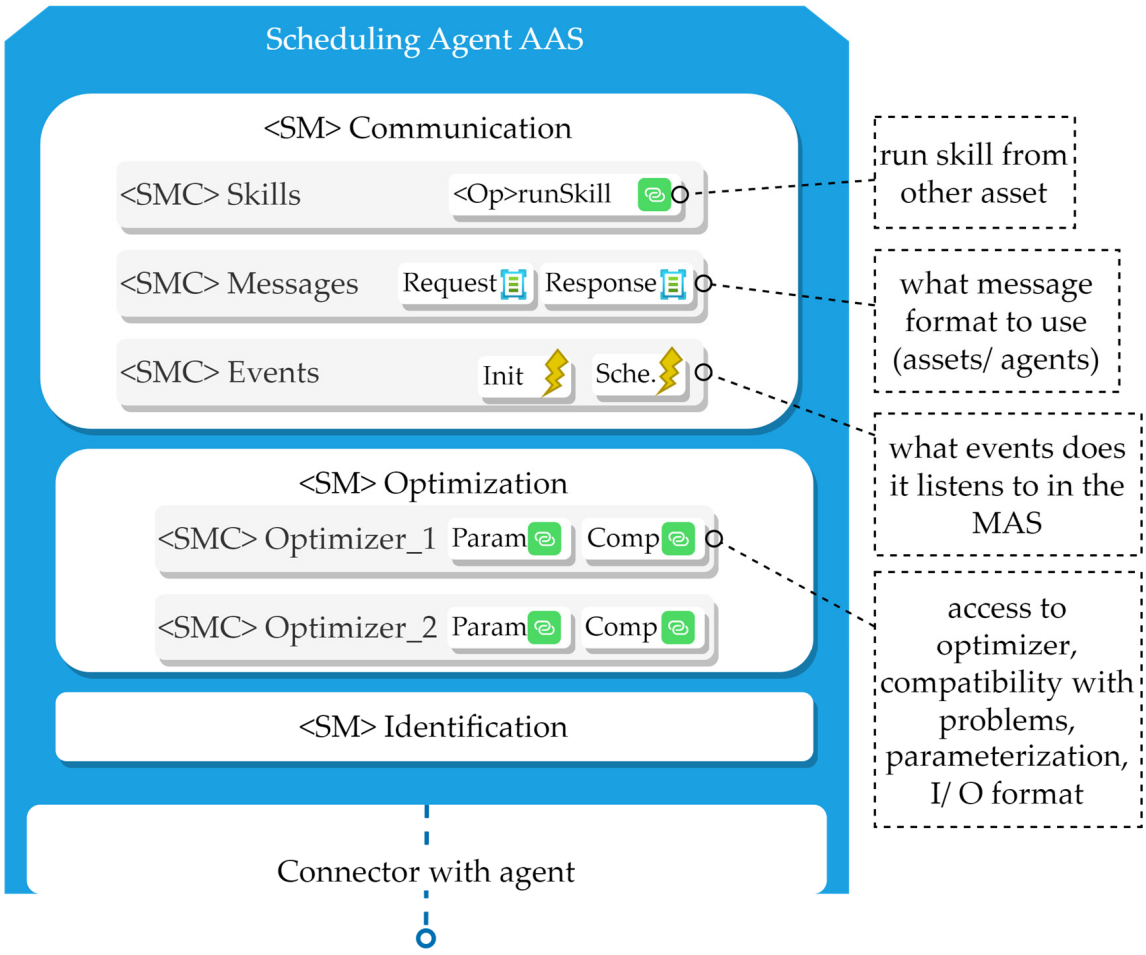


Figure 2. Overview of sub-models (SMs) in scheduling meta-agent AAS.

The "Identification" SM indicates all the required information that distinguishes a specific agent instance from the other agents and specifies information regarding installation and access to the software. More specifically, this module indicates the agent software metadata regarding the platform and the machine that the agent runs and also the agent's specific name and scope. It is not important with regard to the optimization functionalities of the agent, yet it is necessary from the developer's point of view.

The next SM of the agent is the "Optimization" SM. This contains the different optimization services of the agent, i.e., the schedulers/planners, which take care of solving the optimization problems that the agent is requested to. The main challenge of an optimization agent is to avoid case-dependent solutions and achieve generality in the range of problems that can be solved. This can only be achieved if the intelligence of the agent is extended farther than one optimization algorithm or method. This key concept, which was also described in the initial sections, was the core logic behind the meta-agent proposed in the current research. In other words, the authors aimed to provide agent software that has more than one optimization methodology and uses it effectively with respect to the problem it is faced with. In the following figure, the template metamodel of the optimization SM is given for one optimizer example and contains all required information for the agent to access and use the optimizer when required.

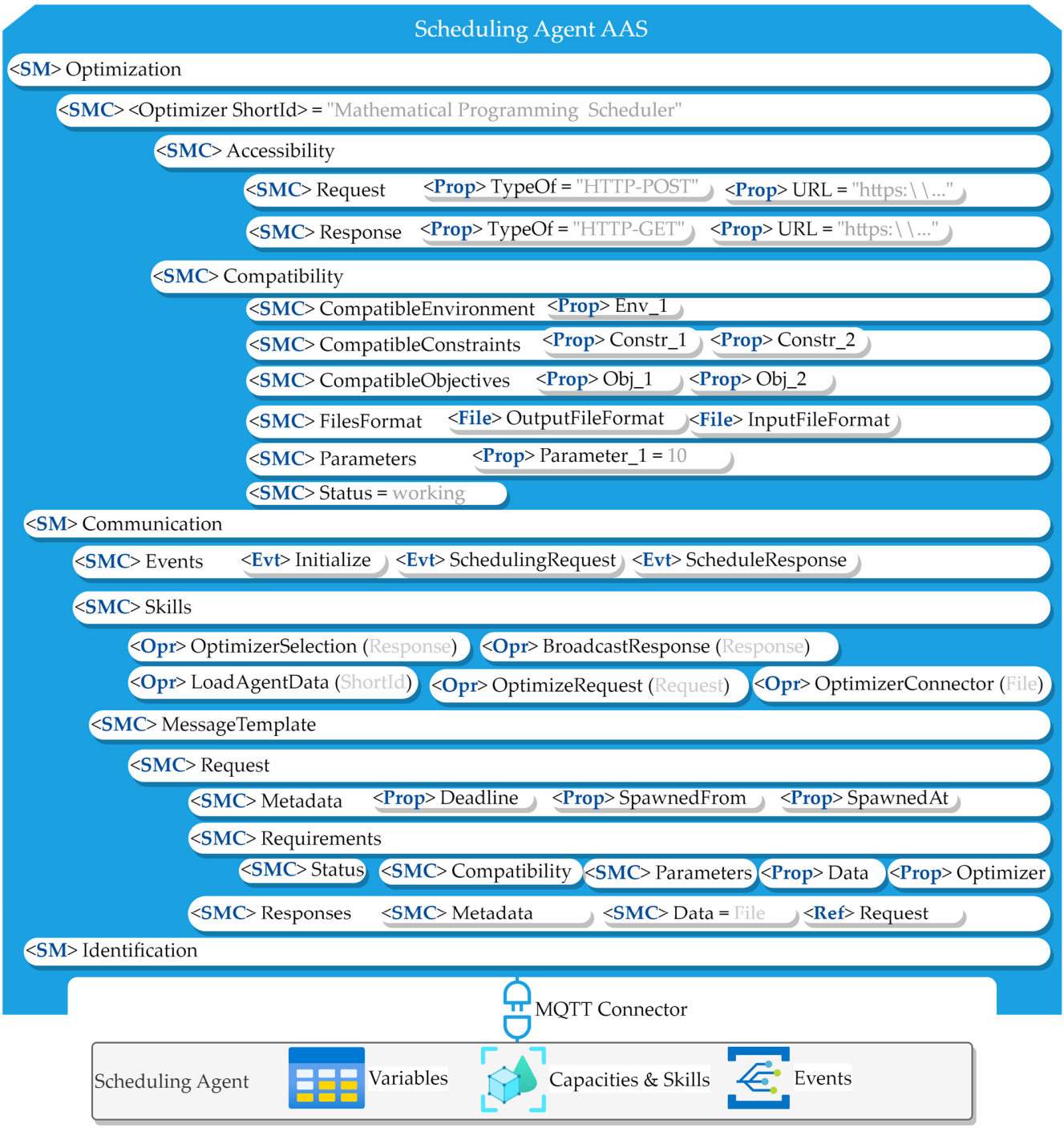


Figure 3. Scheduling agent AAS description.

Within the "Optimization" SM displayed in Figure 3, there are different optimizers, SubModelCollections (SMC), each one representing a different optimization method/algorithm. There are several important aspects to describe each one of the algorithms accurately, the first being accessibility; in other words, where the algorithm can be found and how it can be accessed. Most of the time, the algorithmic part of the optimization is not run within the same resources as the agent, and this is usually performed either for performance or maintenance reasons, meaning that decentralized optimizers are easier to manage when maintenance/update is required (rather than software on the factory site), and computational resources can be empowered as required, or for the practical reason that the algorithm is developed in a different programming framework. In any case, it is important for the agent to

be aware of how the algorithm implementation can be managed and be able to do that not only from a local perspective but also remotely.

The next and most important part with respect to the management of the optimization toolbox is the Compatibility SMC, which allows the agent to identify the cases in which this algorithm/optimizer is suitable for a problem. The description of a scheduling problem can be achieved using three notations. The three notations represent (i) the environment, (ii) the problem constraints, and (iii) the objectives and are enough to classify any schedule optimization problem. Similarly, these notations can be used to describe the sets of scheduling problems that the specific algorithm is compatible with. As an example, for an algorithm that is compatible with Job Shop environments, the constraints of release time and a makespan objective mean that the Job Shop scheduling problem for minimizing the makespan given jobs' release time can be solved. However, in cases where sequence-dependent setup delays are also included, the algorithm would not be able to portrait that constraint and thus, the solution will be practically wrong, and the same applies to any kind of combination of these notations. It is also important to highlight that, in recent research, there are problems that a polynomial time algorithm has been found, or at least to the relaxation of the initial problem. This is useful information for the agent in order to navigate across a database of the different algorithms and select the one that has the best time/memory performance. Of course, in most cases, scheduling problems remain hard to optimize, and polynomial complexity cannot be achieved; in those cases, there can be more than one available algorithm in the list of optimizers, and the selection is part of the agent's logic that has been developed.

The above figure displays the scheduler selection process based on compatibility. As illustrated, after the agent has parsed information from the scheduling request data into the internal format, it analyses the data in order to identify the problem type for the specific dataset. The problem type definition using the three notations described above can be addressed via basic rules for the process plan identifying the different routes that are available for the different jobs and the relations between them. As also highlighted within the later sections of this paper, the production information regarding the process plan, material requirements, work orders, etc., is not something standardized within this research study. In other words, the structure and format of information is an implementation decision, while there are different information frameworks and standards suitable for describing production schedule data.

Following the three-notation modeling mechanism for the scheduling problem allows the agent to characterize any scheduling problem as long as the rules are defined for each specific type. Specifically, the agent needs to be able to identify, given the input data, which rules need to be applied in order to define whether the data are derived from a specific type of problem. To give a more practical example, Job Shop environments, for example, are supported by the condition that there is only one alternative route for each order, in contrast to the Flexible Job Shop, which contains more than one. Another example is the case of batch versus continuous manufacturing, which can be defined by the ability of the scheduler to include job release time or/and family of jobs, which can be linked with material arrival dates and product types. There are, however, even more complex examples, such as inventory constraints and case-dependent policy constraints, which are hard to classify into a specific cluster of problems. From the implementation side, however, the agent is able to analyze any given problem into these categories as long as it is given a detailed set of rules that need to be satisfied for this class.

Finally, the performance of the algorithm was another aspect that needed to be taken into account by the agent in cases where there is more than one compatible algorithm for the problem. This aspect was addressed individually by each scheduler, which provided a specific estimation of time and accuracy based on the scale of the problem. This can be addressed by using a linear interpolation among existing experimental data and empirical results. This aspect is described in an abstract manner, allowing the developer to define its own functions for returning an estimation of time and accuracy for the un-

derlying algorithm. On top of that, in cases where the algorithm contained some kind of hyper-parameterization, a function can be defined to link the requirements in time or/and accuracy with the parameters to be used for the algorithm.

As long as the scheduling method is selected, the I/O format used to load the problem to the algorithm's implementation and receive the output should be defined. This is a design decision for any given software, and to achieve interoperability, the agent should be able to parse the information in the given format as it is the language of communicating with the algorithm. The files will pass to the accessibility endpoints using the correct protocol, and thus, a connection will be established. Let us, for example, have a specific version of an algorithm being developed on a computational resource, which is then made public using a REST API service where new models can be loaded directly to the algorithm through an external source in a specific XML format file. In that case, the agent would need to serialize the scheduling information into the specific XML format and then post it to the REST service, which will trigger the algorithm. The same logic can be applied in any given file format and communication protocols, given that the agent is programmed to do so.

The next and equally important SM is "Communication", which establishes all the necessary information for the intercommunication of the agent with the rest of the MAS. In essence, AOP is based on events, capacities, and skills to define the agent's workflow. As such, the first important component is defining the events which the agent is programmed to listen to within the MAS. The first primary event is the initialization one, which is part of any agent object and is called immediately when the agent is spawned by the system. The agent would have a universal unique ID, which allows the software to spot the specific AAS associated with that ID. The AAS metamodel will configure some of the internal data and connections of the agent and will set the agent as active. At this point, the agent is functional, and its primary behavior is to listen, broadcast, and react to events from/to the MAS. The system may produce a wide variety of events; however, each agent will react to only a specific subset of events that are associated with the scheduling operations of the system. The scheduling meta-agent specifically listens to the "SchedulingRequest", which can be spawned by another agent requesting the schedule optimization for a set of production data. The request is modeled in greater detail by the "Request" SMC containing metadata information on top of the data defining the scheduling problem, i.e., the "Data" file. At this point, it is also important to highlight that the format of the scheduling information is another important aspect for the agent as it will allow the creation of context out of a stream of data that is provided. Formatting production information, however, is a topic of its own, with various abstract models and standards provided (e.g., ISA-95) in order to encapsulate information in a generic and standardized manner. This is out of the scope of this paper, as it focuses on modeling the agent's interaction and capabilities; nevertheless, within the conclusion section, a focus is particularly given to future works and limitations over that aspect.

The compatibility requirements are the problem description information, namely environment, constraints, and objectives, which allow the agent to map the suitable algorithms to the specific problem description. Equivalently, the parameters may override any existing default parameterization of the algorithm performed during the initialization of the agent. The response is created as the optimization process has finished, while the "ScheduleResponse" is also broadcasted.

Now, an interesting debate on the utilization of AAS is whether dynamic information should be stored on the AAS directly. In essence, the AAS file is an asset description metamodel, and it is not meant for direct storage of asset values. In other words, in the case of a shopfloor machine, an AAS would enable access to machine-specific components, e.g., a machine sensor. However, it does not necessarily mean that data would also be loaded upon the AAS repository, i.e., uploading sensor data upon the AAS file. Most of the time, this is a design choice, as the description of the AAS does not exclude that dynamic data may also be loaded. On the other hand, AASs do not act as an extended storage of historical data, and although they could perform this role, it is usually out of the scope of using the

AAS. With that being said, there can be defined two separate methods for requesting and receiving a new schedule operation in the planning agent (shown in Figure 4). The first and more orthological way is through the MAS, based on the principles of AOP. The agent is, in essence, an entity that responds to events from the MAS, such that the planning agent is an entity that responds to the scheduling events which are broadcasted by other agents. A significant detail is that the scheduling request sent from the agent should follow the format of the request that is referenced within the AAS description. So, in practice, the inner format of the event follows the same object format as the SMC described within the AAS, except it is transferred as a serialized object because it is sent directly as a message within the MAS. The scheduling agent can follow the procedure—of selecting the optimizer and optimizing the schedule—for completing the scheduling request and uploading the input message onto the AAS repository for traceability reasons, if needed.

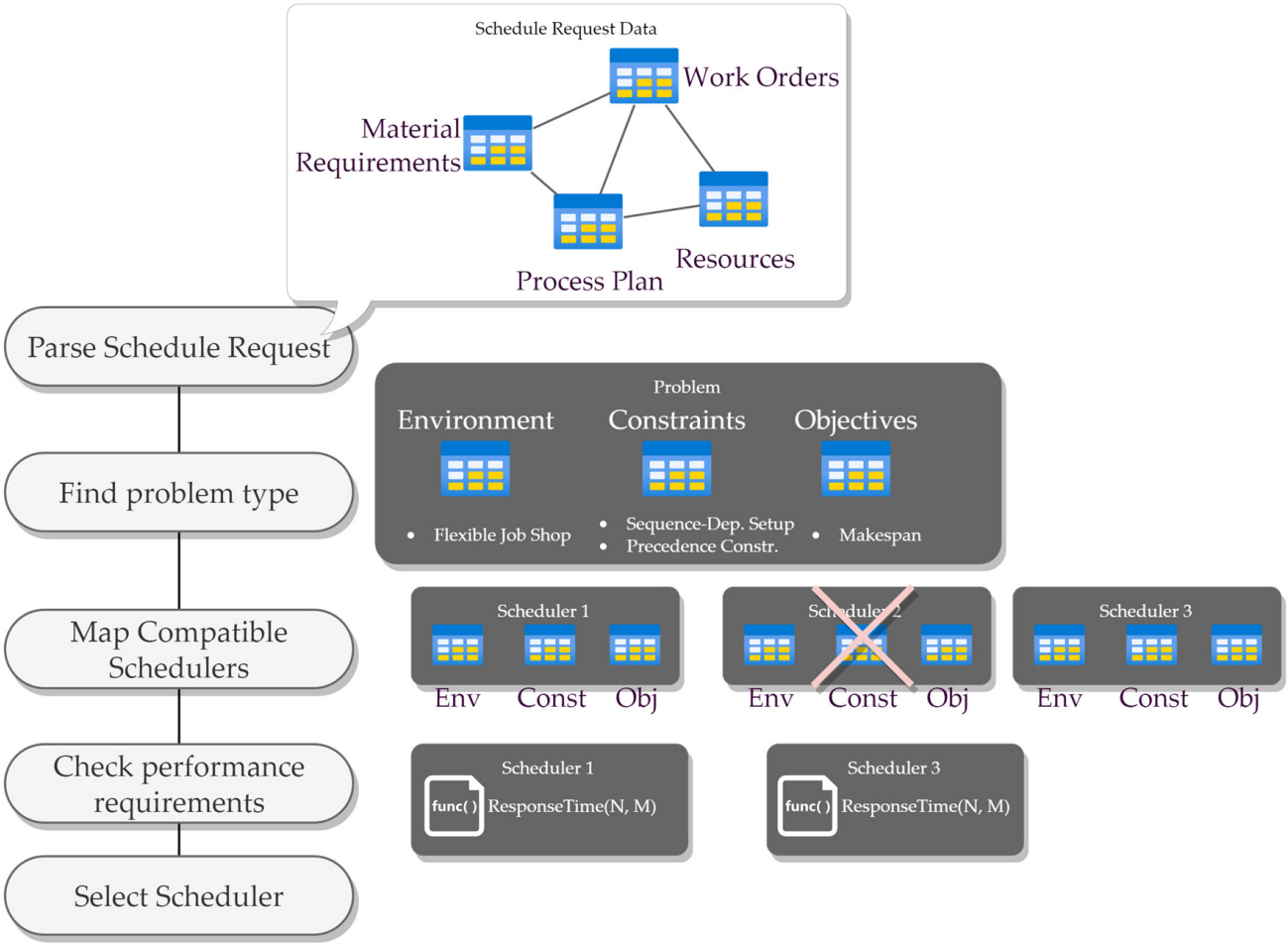


Figure 4. Workflow for scheduler selection within the agent.

The second way to request a new schedule is directly via the AAS. In practice, this is what AAS files are for: providing access to the asset's operations and information free of hardware or software-specific requirements. In the case of the agent, this may slightly differ, as agents primarily interact with each other using AOP; nevertheless, this method also allows external entities (out of the MAS) to interfere with the agent when required. Just by calling the corresponding operation from the agent's skill SMC and uploading the accompanying information, any external software can have access to the scheduling services provided by the agent. This type of interaction is primarily for third-party entities outside of the MAS, which have no access to the message-based communication established by the MAS framework.

4. Industrial Implementation

Since agents are programmed based on AOP, an MAS framework implementation is required in order for these entities to be spawned and managed. In this case, they utilize the Janus framework and SARL programming language. There are four main types within AOP, namely, agents, events, capacities, and skills. The agent is the equivalent of a class in OOP and is a placeholder for the methods, events, skills, and variables that consist of the agent as an entity. Capacity, on the other hand, is a generic description of an agent's capability and is independent of the implementation. As an example, a capacity may be a pick and place in the case of a robot control agent, whereas the skill, on the other hand, is the specific implementation and, in this case, the set of step commands needed to perform the pick and place operation. Lastly, the events are the way that the agents interact, and this is what makes it different from OOP. Events are broadcasted within the network of agents by the agents unaware of who will be the listener of the events. In other words, the agents are autonomous entities that receive an event as long as the event is compatible with their skills and internal operations and may or may not produce new events for the system.

In JANUS and the proposed implementation, there are four main concepts as well, described as follows:

- Agent: An agent is capable of receiving the scheduling request from the UI and selecting the corresponding scheduling algorithm to perform the scheduling calculation. After the scheduling output has been completed, the planning agent emits the schedule response to other agents in the MAS, who will handle this information based on their functionality as well.
- Skills and capacity: A capacity is a wrapper of skills. The reason for having a capacity is that an agent can implement a skill from a certain capacity with a specific parameterization. In the planning agent implementation, the skills refer to the different capabilities of the planning agent. The planning agent is capable of choosing the correct scheduler, performing the scheduling task through scheduler interfaces and REST API connections, and posting the scheduling task outcome to a UI.
- Events: There are two kinds of events in this implementation. The schedule request, where another entity in the MAS broadcasts it, and the schedule response, where the planning agent broadcasts when the scheduling task has finished.

These types were implemented into Janus, as displayed in Figure 5. SARL is built on top of the Java programming language, which makes it extremely compatible with the exploitation of Java interfaces, methods, and classes to perform actions for which Java has existing libraries and packages. In addition to the traditional JANUS entities, some additional classes were implemented:

- UIConnector: When the scheduler finishes its operation, the outcome needs to be shown in a UI, where the user will be able to read it and decide the following production activities (apply or reject the schedule).
- SchedulerInterface: This class is used for the preprocessing and post-processing. The input needs to be reformed into the proper scheduler input format, and the output from the scheduler needs to be reformed into a readable output.
- SchedulersConnection: The schedulers listen to the schedule request through a REST API and provide the response with the use of MQTT. Thus, an additional class for the connection of the schedulers' endpoints with the planning agents' skills was required.

The planning agent contained (was able to receive/listen to) three different MAS events:

- "Initialization", spawn of the agent and configuration of the connections with the schedulers based on the AAS;
- "Scheduling request", a message containing the production data to be scheduled;
- "Schedule response", a chain event broadcasted after the end of the scheduling process.

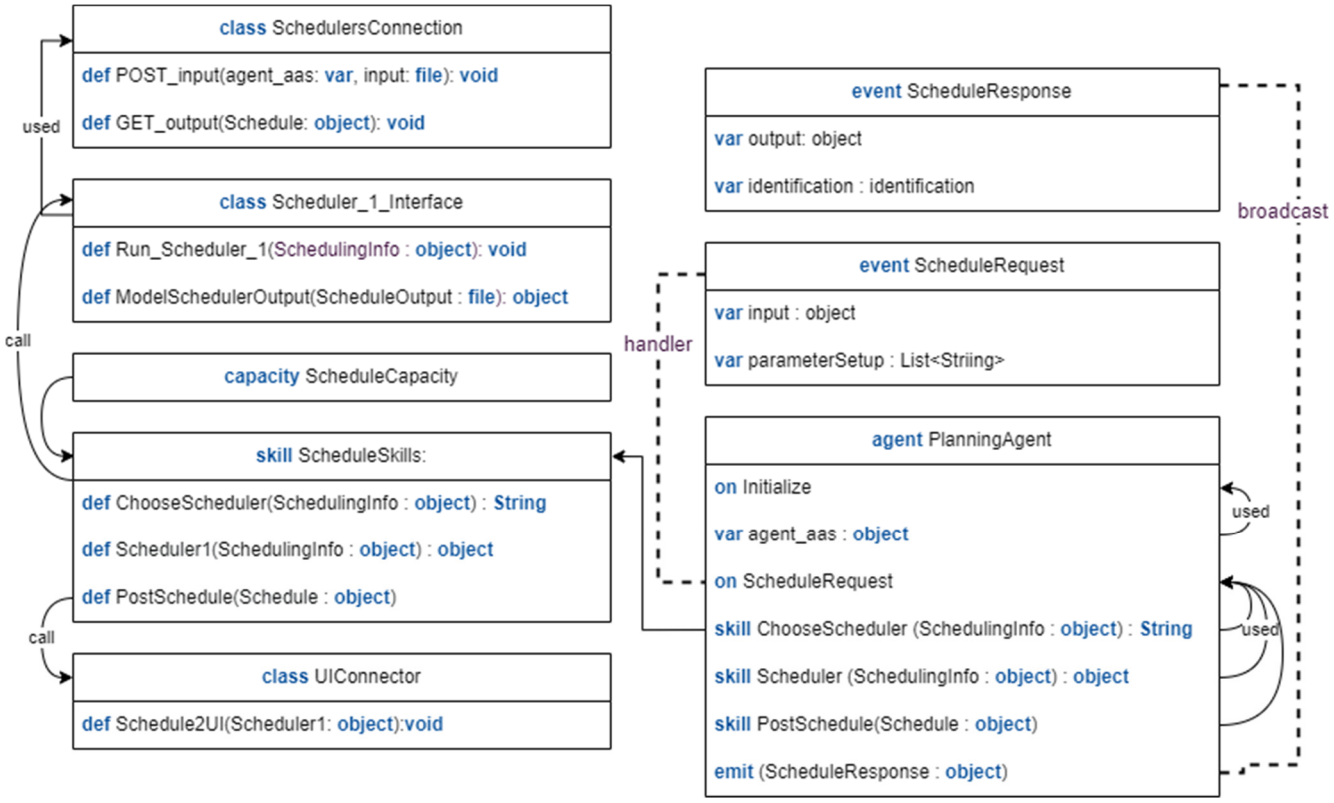


Figure 5. Class diagram implementation of agent in Janus.

The scheduling request followed the structure defined within the agent's AAS and contained the information for the scheduling problem that needed to be optimized. Before moving to the optimization, the agent should make the decision of which optimizer/scheduler to use. As long as the agent had already defined the connections with the schedulers and had received a collection of environment, constraints, and objectives that were compatible with each algorithm, it needed to classify the problem and match it with any compatible schedulers. The scheduler selection algorithm implemented a specific set of rules linked to each type of environment and constraints over the production data and assigned the corresponding notations to the scheduling request. Once this process was performed, the agent excluded the non-compatible algorithms and selected one of the remaining ones. Now, the process for selecting among the remaining algorithms had to do with performance. In this case, each algorithm had a parameterized method based on the number of resources and jobs and returned an estimation of the scheduling time required. The agent would select the closest to the demand given by the request. In cases where there was no demand, all the schedulers would be used simultaneously, and multiple scheduling responses would be produced.

A similar procedure could take place for scheduling requests coming from outside the MAS. The AAS platform, rather than hosting the AAS elements' values, also provided endpoints for the AAS operations. Any third-party components, in this case, the UI, could request a new schedule by posting the request over the AAS platform on the operation public URL. An MQTT connection between the AAS and the agent in Janus was established, and the same process would be followed in order to be able to broadcast the response back to the UI.

The different software components are displayed in Figure 6, showing the software and hardware deployment architecture for industrial implementation, including the hardware devices (servers) utilized for the specific industrial case. The shopfloor UI, as the name suggests, should be deployed on the production line, allowing the user to interact with the MAS and display the results. The shopfloor UI was a Java web application installed locally

on the shopfloor computer on the factory site, while it was integrated with information coming from the ERP database. The connection between the ERP DB and the UI was made using SQL queries, loading all the production orders that are to be produced within the upcoming days. The AAS platform was also installed on the local production network, holding information about the agents deployed for the specific use case. During the initialization, the MAS will recall these AAS files in order to deploy the different scheduling agents and define the connection between the agents and the schedulers. The MAS was also installed on a Virtual Machine (VM) on the local network. The algorithms/schedulers, however, were deployed on a different server outside of the industrial network. The specific VM required the highest specifications for memory and performance as it contained the optimization processes of the framework. It used 32 GB of RAM and an Intel (R) Core (TM) i7-10700 CPU @ 2.90 GHz.

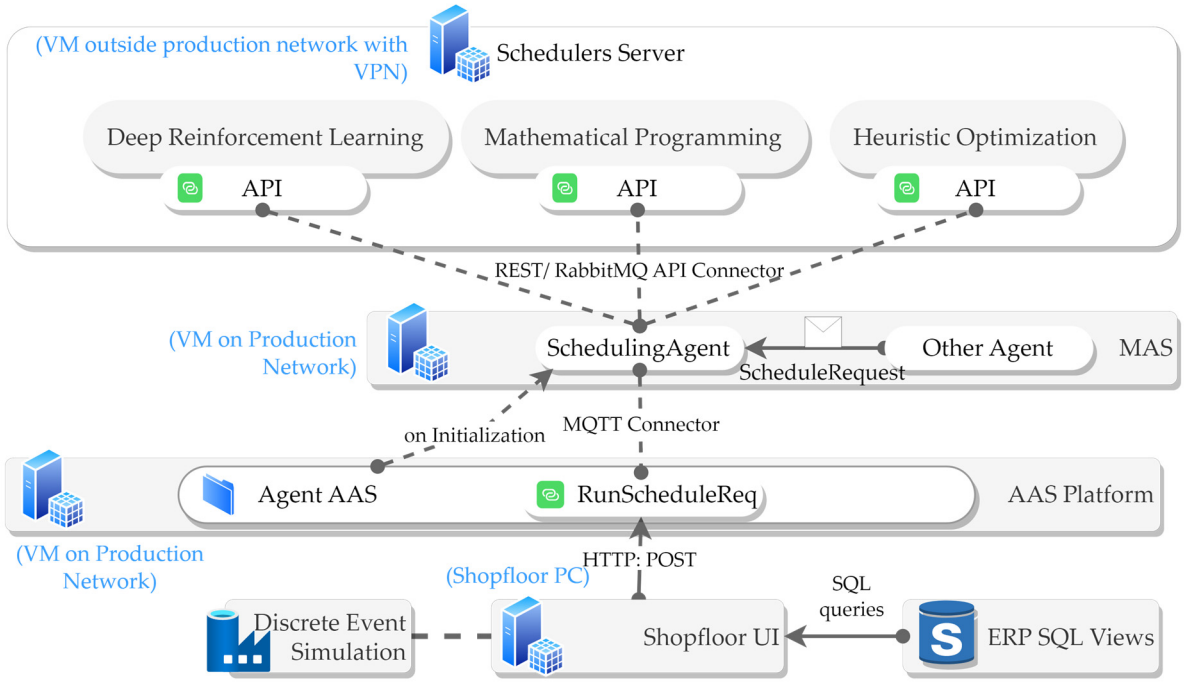


Figure 6. Software and hardware deployment architecture for industrial implementation.

The industrial case came from a bicycle Original Equipment Manufacturing (OEM) industry specifically for the departments of painting and wheel assembly. Three different schedulers were deployed: (1) a heuristic multi-objective scheduling framework, (2) a Mixed Integer Programming [68] (MIP) model optimizer for production scheduling, and (3) a deep reinforcement learning (DRL) scheduler for dynamic production scheduling. For the latter two (2, 3), two Flask applications were deployed to integrate the algorithms as REST services. Similarly, for the heuristic scheduler, a Java REST service was developed. On top of that, a RabbitMQ broker was used in order to return information regarding the status of the optimization process.

The heuristic algorithm was used both for the painting and the wheel assembly lines' allocation of orders, while the MIP and DRL were used individually for the painting and wheel assembly, respectively. The heuristic algorithm contained three adjustable hyperparameters, which drove the search process and could affect the performance in time and accuracy. Similarly, for the MIP optimization, the MIP gap and the threads were two adjustable parameters, while for the DRL, two separate models were trained with different accuracy results.

On top of that, two separate Discrete Event Simulation (DES) models were developed for the two departments in order to run the scheduling results and display the performance indicators to the user. The DES models were separate from the MAS and were connected

to the UI application. This allowed the user to perform multiple scheduling requests, giving different optimization requirements for the agent after validating the results on the DES models. LANNER WITNESS HORIZON was used for the DES model development. The user was also able to send scheduling operations to the planning agent in JANUS in an abstract way without the need to specify the corresponding problem, while the agent's logic was able to break down the problem and allocate it to the corresponding scheduling algorithms that were compatible. Seamless integration between the SARL software and the individual schedulers was achieved while events were exchanged between the production simulation and the agent. In Figure 7, the DES tab in the UI is shown, where the user can identify some production KPIs (buffers and resources capacity, resources utilization, production makespan in mins, etc.), with the resulting schedule applied in the corresponding DES model of the production system. After the application of different workloads for different departments, the results indicated that the planning agent was able to distinguish between different well-defined problems and utilize the correct scheduling tools, respectively. This provided benefits for the user, as it automated the optimization process and potentially utilized the most suitable algorithms for each problem.

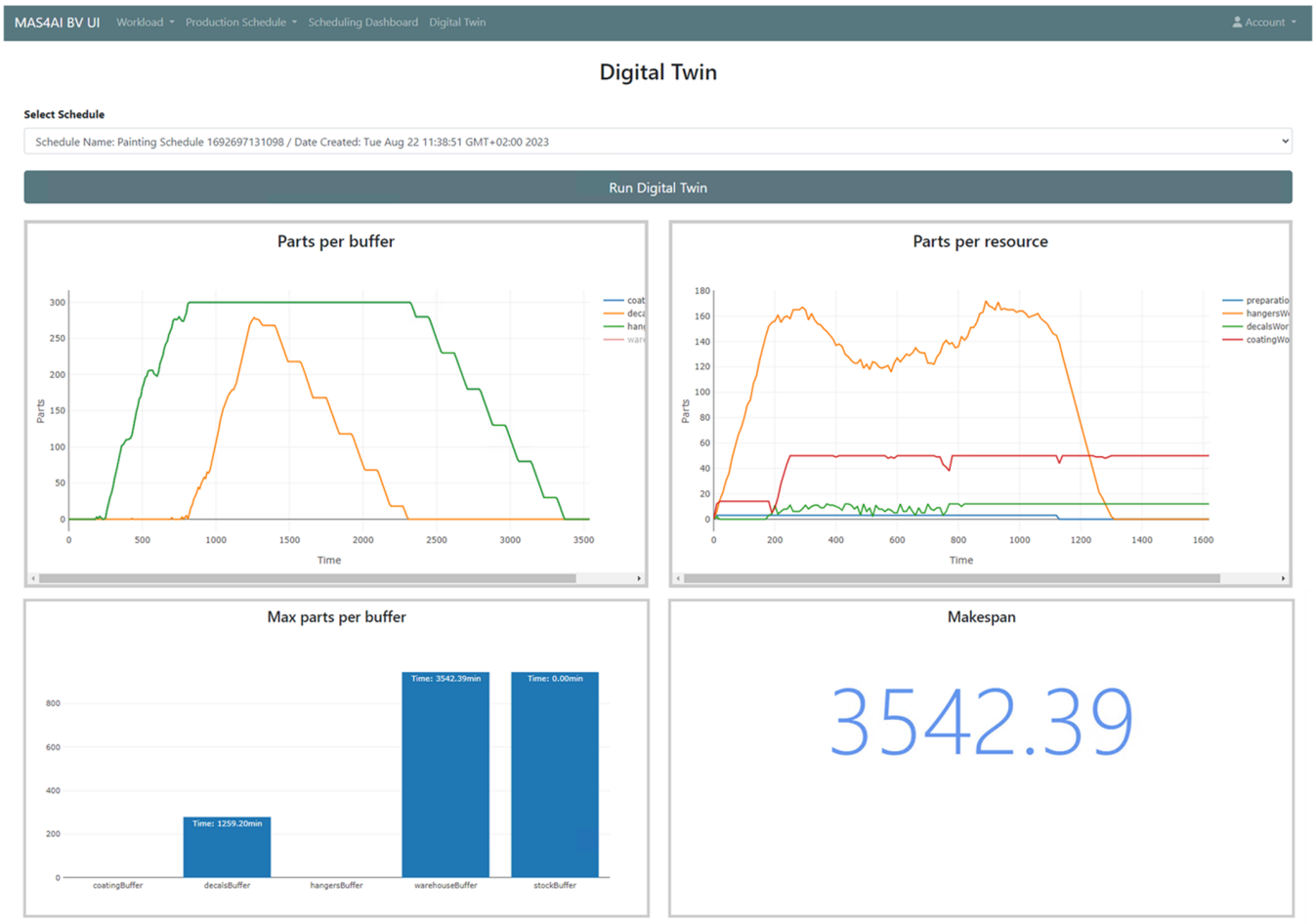


Figure 7. Screenshot from UI tab for resulted schedule evaluation with the use of DES.

5. Conclusions and Future Work

This work focuses on the design and implementation of a scheduling agent using the AAS concept. The primary objective was to enable a generic agent model capable of supporting multiple scheduling methods and exploiting them with respect to the corresponding problem. The proposed AAS model allows the agent interactions to be described in an abstract way, enables interoperability with external software via AAS platform-hosted API, and, finally, provides a description of the agent's capability. The research aimed to

highlight a way to model the capabilities of scheduling algorithms and their compatibility with different problems based on three basic notations. Concentrating on an adaptable scheduling agent approach, this method allows the agent to classify problems based on these three notations by applying rules over the provided information. As such, the agent may be able to create context out of the input information and selectively allocate a problem to a specific algorithm. The proposed approach, in addition, seems promising on the scalability aspect that may enable the addition of an arbitrary set of scheduling methods and compatible environments along with the specific classification rules for the agent. The proposed information model and agent architecture were implemented within the Janus open-source framework for developing MAS in order to establish the basic agent, capacities, skills, and event format for its deployment within an MAS.

There are some fundamental differences in using the AAS concept for an agent versus other types of assets. The differences mainly occur because agents are already supported by the AOP language for exchanging information, and this is, in essence, another communication language different from the AAS. As such, this work proposes the establishment of the message-based interaction within the MAS as well as provides communication functionalities with other systems that were not built on top of AOP and are not able to participate within the decentralized communication framework of the agents.

In the current research work, agents are described mainly on the optimization and communication functionalities, and it aims to allow the system to identify the capabilities that the agent can provide and thus redirect the scheduling request to the right algorithm. This was supported by the description of the scheduling problem based on three notations, namely the environment, constraints, and objectives, which is a widely adopted practice for production scheduling problems. The framework can, in this way, be scalable to different problems as long as the agent is able to make a rule-based classification of the input data for each problem type. This may boost its applicability in different industrial scenarios, providing the ability to implement different scheduling algorithms and services in a single scheduling agent architecture.

Future work will focus on establishing a common information format that will be used to exchange production schedule-related information to be more abstract and does not depend on the system's ability to read a custom data model that may be applied only in individual industrial use cases. Nowadays, although the aspects of accessibility and compatibility to define where and how a system can exchange information and request services from the scheduling agent are specified, the specific way in which information should be structured within the message has not yet been defined.

Author Contributions: Conceptualization, V.S., N.N. and K.A.; methodology, V.S., E.B. and P.M.; software, V.S.; validation, V.S., E.B. and P.M.; resources, N.N. and K.A.; writing—original draft preparation, V.S., E.B. and P.M.; writing—review and editing, N.N. and K.A.; visualization, V.S.; supervision, K.A.; project administration, N.N. and K.A.; funding acquisition, K.A. All authors have read and agreed to the published version of the manuscript.

Funding: This project has received funding from the European Union's Horizon 2020 research and inno-vation program under grant agreement No 957204 MAS4AI. The dissemination of results herein reflects only the authors' view, and the Commission is not responsible for any use that may be made of the information it contains.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to privacy.

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

References

1. Chryssolouris, G. *Manufacturing Systems: Theory And Practice*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2006.
2. Bader, S.; Barnstedt, E.; Bedenbender, H.; Berres, B.; Billmann, M.; Ristin, M. *Details of the Asset Administration Shell—Part 1: The Exchange of Information between Partners in the Value Chain of Industrie 4.0 (Version 3.0RC02)*; Federal Ministry for Economic Affairs and Climate Action (BMWK): Berlin, Germany, 2022. [CrossRef]
3. Platis, D. Software Development Bot Ecosystems. 2021. Available online: https://hdl.handle.net/20.500.12380/303876 (accessed on 9 March 2023).
4. O'Hare, G.; Jennings, N. *Foundations of Distributed Artificial Intelligence*; John Wiley & Sons: Hoboken, NJ, USA, 1996.
5. Oliveira, E.; Fischer, K.; Stepankova, O. Multi-agent systems: Which research for which applications. *Robot. Auton. Syst.* **1999**, *27*, 91–106. [CrossRef]
6. Chen, F.; Ren, W. On the control of multi-agent systems: A survey. *Found. Trends®Syst. Control.* **2019**, *6*, 339–499. [CrossRef]
7. McArthur, S.D.J.; Davidson, E.M.; Catterson, V.M.; Dimeas, A.L.; Hatziargyriou, N.D.; Ponci, F.; Funabashi, T. Multi-agent systems for power engineering applications—Part II: Technologies, standards, and tools for building multi-agent systems. *IEEE Trans. Power Syst.* **2007**, *22*, 1753–1759. [CrossRef]
8. Brazier, F.M.T.; Dunin-Keplicz, B.M.; Jennings, N.R.; Treur, J. Desire: Modelling multi-agent systems in a compositional formal framework. *Int. J. Coop. Inf. Syst.* **1997**, *6*, 67–94. [CrossRef]
9. Iglesias, C.A.; Garijo, M.; González, J.C.; Velasco, J.R. Analysis and design of multiagent systems using MAS-CommonKADS. In *ATAL 1997—Intelligent Agents IV Agent Theories, Architectures, and Languages*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 1997; Volume 1365, pp. 313–327. [CrossRef]
10. Lidula, N.; Rajapakse, A. Microgrids research: A review of experimental microgrids and test systems. *Renew. Sustain. Energy Rev.* **2011**, *15*, 186–202. [CrossRef]
11. Ghadimi, P.; Ghassemi Toosi, F.; Heavey, C. A multi-agent systems approach for sustainable supplier selection and order allocation in a partnership supply chain. *Eur. J. Oper. Res.* **2018**, *269*, 286–301. [CrossRef]
12. Calegari, R.; Ciatto, G.; Mascardi, V.; Omicini, A. Logic-based technologies for multi-agent systems: A systematic literature review. *Auton. Agents Multi-Agent Syst.* **2020**, *35*, 1. [CrossRef]
13. Zia, M.F.; Elbouchikhi, E.; Benbouzid, M. Microgrids energy management systems: A critical review on methods, solutions, and prospects. *Appl. Energy* **2018**, *222*, 1033–1055. [CrossRef]
14. Sujil, A.; Verma, J. *Multi Agent System: Concepts, Platforms and Applications in Power Systems*; Springer: Berlin/Heidelberg, Germany, 2018; Volume 49.
15. Gul, F.; Mir, A.; Mir, I.; Mir, S.; Islaam, T.U.; Abualigah, L.; Forestiero, A. A Centralized Strategy for Multi-Agent Exploration. *IEEE Access* **2022**, *10*, 126871–126884. [CrossRef]
16. Folino, G.; Forestiero, A.; Spezzano, G. A Jxta based asynchronous peer-to-peer implementation of genetic programming. *J. Softw.* **2006**, *1*, 12–23. [CrossRef]
17. Hu, B.; Guo, W.; Jing, Z. Electric vehicles operation simulation system based on multi-agent system. In Proceedings of the 2014 IEEE Conference and Expo Transportation Electrification Asia-Pacific (ITEC Asia-Pacific), Beijing, China, 31 August–3 September 2014; pp. 1–5. [CrossRef]
18. Dou, C.-X.; Liu, B. Multi-agent based hierarchical hybrid control for smart microgrid. *IEEE Trans. Smart Grid* **2013**, *4*, 771–778. [CrossRef]
19. Aung, H.N.; Khambadkone, A.M.; Srinivasan, D.; Logenthiran, T. Agent-based intelligent control for real-time operation of a microgrid. In Proceedings of the 2010 Joint International Conference on Power Electronics, Drives and Energy Systems & 2010 Power India, New Delhi, India, 20–23 December 2010; pp. 1–6. [CrossRef]
20. Ansari, J.; Gholami, A.; Kazemi, A. Holonic structure: A state-of-the-art control architecture based on multi-agent systems for optimal reactive power dispatch in smart grids. *IET Gener. Transm. Distrib.* **2015**, *9*, 1922–1934. [CrossRef]
21. Li, T.; Xiao, Z.; Huang, M.; Yu, J.; Hu, J. Control system simulation of microgrid based on IP and Multi-Agent. In Proceedings of the 2010 International Conference on Information, Networking and Automation (ICINA), Kunming, China, 18-19 October 2010; Volume 1, pp. V1-235–V1-239. [CrossRef]
22. Pipattanasomporn, M.; Feroze, H.; Rahman, S. Multi-agent systems in a distributed smart grid: Design and implementation. In Proceedings of the IEEE/Pes Power Systems Conference & Exposition, Seattle, WA, USA, 15–18 March 2009; pp. 1–8. [CrossRef]
23. Xiao, Z.; Li, T.; Huang, M.; Shi, J.; Yang, J.; Yu, J.; Wu, W. Hierarchical MAS based control strategy for microgrid. *Energies* **2010**, *3*, 1622–1638. [CrossRef]
24. Amor, M.; Fuentes, L. Malaca: A component and aspect-oriented agent architecture. *Inf. Softw. Technol.* **2009**, *51*, 1052–1065. [CrossRef]
25. Gutknecht, O.; Ferber, J. The MadKit agent platform architecture. In *AGENTS 2000—Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*; Springer: Berlin/Heidelberg, Germay, 2001; pp. 48–55. [CrossRef]
26. Redjimi, K.; Redjimi, M. A Multi-agent system for industrial simulators design. In *Advances in Deep Learning, Artificial Intelligence and Robotics*; Springer: Cham, Switzerland, 2022; pp. 129–140. [CrossRef]
27. Michel, F. MaDKit. 2021. Available online: https://hal-lirmm.ccsd.cnrs.fr/lirmm-03833158 (accessed on 9 June 2023).

28. Gaud, N.; Galland, S.; Hilaire, V.; Koukam, A. An organisational platform for holonic and multiagent systems. In *ProMAS 2008—Programming Multi-Agent Systems*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 104–119. [CrossRef]
29. Galland, S.; Knapen, L.; Yasar, A.-U.; Gaud, N.; Janssens, D.; Lamotte, O.; Koukam, A.; Wets, G. Multi-agent simulation of individual mobility behavior in carpooling. *Transp. Res. Part C Emerg. Technol.* **2014**, *45*, 83–98. [CrossRef]
30. Chaudhry, Q.A. An introduction to agent-based modeling modeling natural, social, and engineered complex systems with NetLogo: A review. *Complex Adapt. Syst. Model.* **2016**, *4*, 11. [CrossRef]
31. Nwokoye, C.H.; Mbeledogu, N.N.; Paul, R.U.; Ugwunna, C. Complementing malware epidemic agent-based models with routing protocols of communication networks using NetLogo. In Proceedings of the Information and Communication Technology for Competitive Strategies (ICTCS 2022), Chandigarh, India, 9–10 December 2022; Springer: Singapore, 2022; pp. 833–847. [CrossRef]
32. Sánchez, A.; Villarrubia, G.; Zato, C.; Rodríguez, S.; Chamoso, P. A gateway protocol based on fipa-acl for the new agent platform pangea. In *Trends in Practical Applications of Agents and Multiagent Systems*; Advances in Intelligent Systems and Computing; Springer: Cham, Switzerland, 2013; pp. 41–51. [CrossRef]
33. Leon, F.; Paprzycki, M.; Ganzha, M. A Review of Agent Platforms. Available online: http://jacamo.sourceforge.net/?page_id=19 (accessed on 9 March 2023).
34. Collier, N.; North, M. *Repast HPC: A Platform for Large-Scale Agent-Based Modeling*; John Wiley & Sons: Hoboken, NJ, USA, 2011; pp. 81–109. [CrossRef]
35. Borshchev, A. Multi-method modelling: AnyLogic. In *Discrete-Event Simulation and System Dynamics for Management Decision Making*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 2014; pp. 248–279. [CrossRef]
36. Li, X.; Sun, Y. Stock intelligent investment strategy based on support vector machine parameter optimization algorithm. *Neural Comput. Appl.* **2019**, *32*, 1765–1775. [CrossRef]
37. Cossentino, M.; Gaud, N.; Hilaire, V.; Galland, S.; Koukam, A. ASPECS: An agent-oriented software process for engineering complex systems. *Auton. Agents Multi-Agent Syst.* **2009**, *20*, 260–304. [CrossRef]
38. SARL Agent-Oriented Programming Language. Available online: http://www.sarl.io/ (accessed on 9 March 2023).
39. Janus Agent and Holonic Platform. Available online: http://www.sarl.io/runtime/janus/ (accessed on 9 March 2023).
40. Tantik, E.; Anderl, R. Integrated data model and structure for the asset administration shell in Industrie 4.0. *Procedia CIRP* **2017**, *60*, 86–91. [CrossRef]
41. McArthur, S.D.J.; Davidson, E.M.; Catterson, V.M.; Dimeas, A.L.; Hatziargyriou, N.D.; Ponci, F.; Funabashi, T. Multi-agent systems for power engineering applications—Part I: Concepts, approaches, and technical challenges. *IEEE Trans. Power Syst.* **2007**, *22*, 1743–1752. [CrossRef]
42. Nair, A.S.; Hossen, T.; Campion, M.; Selvaraj, D.F.; Goveas, N.; Kaabouch, N.; Ranganathan, P. Multi-agent systems for resource allocation and scheduling in a smart grid. *Technol. Econ. Smart Grids Sustain. Energy* **2018**, *3*, 15. [CrossRef]
43. Aydin, M.E.; Fogarty, T.C. A simulated annealing algorithm for multi-agent systems: A job-shop scheduling application. *J. Intell. Manuf.* **2004**, *15*, 805–814. [CrossRef]
44. Bertolini, M.; Mezzogori, D.; Neroni, M.; Zammori, F. Machine learning for industrial applications: A comprehensive literature review. *Expert Syst. Appl.* **2021**, *175*, 114820. [CrossRef]
45. Cadavid, J.P.U.; Lamouri, S.; Grabot, B.; Pellerin, R.; Fortin, A. Machine learning applied in production planning and control: A state-of-the-art in the era of industry 4.0. *J. Intell. Manuf.* **2020**, *31*, 1531–1558. [CrossRef]
46. Jungbluth, S.; Hermann, J.; Motsch, W.; Pourjafarian, M.; Sidorenko, A.; Volkmann, M.; Zoltner, K.; Plociennik, C.; Ruskowski, M. Dynamic replanning using multi-agent systems and asset administration shells. In Proceedings of the 2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA), Stuttgart, Germany, 6–9 September 2022; pp. 1–8. [CrossRef]
47. Sakurada, L.; Leitao, P.; De La Prieta, F. Engineering a Multi-agent systems approach for realizing collaborative asset administration shells. In Proceedings of the 2022 IEEE International Conference on Industrial Technology (ICIT), Shanghai, China, 22–25 August 2022; pp. 1–6. [CrossRef]
48. Sakurada, L.; Leitao, P.; De la Prieta, F. Agent-based asset administration shell approach for digitizing Industrial assets. *IFAC-PapersOnLine* **2022**, *55*, 193–198. [CrossRef]
49. Komesker, S.; Motsch, W.; Popper, J.; Sidorenko, A.; Wagner, A.; Ruskowski, M. Enabling a multi-agent system for resilient production flow in modular production systems. *Procedia CIRP* **2022**, *107*, 991–998. [CrossRef]
50. Abdel-Aty, T.A.; Negri, E.; Galparoli, S. Asset administration shell in manufacturing: Applications and relationship with digital twin. *IFAC-PapersOnLine* **2022**, *55*, 2533–2538. [CrossRef]
51. Ye, X.; Yu, M.; Song, W.S.; Hong, S.H. An asset administration shell method for data exchange between manufacturing software applications. *IEEE Access* **2021**, *9*, 144171–144178. [CrossRef]
52. Status Report Reference Architecture Model Industrie 4.0 (RAMI4.0). 2015. Available online: www.vdi.de (accessed on 22 December 2022).
53. Wagner, C.; Grothoff, J.; Epple, U.; Drath, R.; Malakuti, S.; Grüner, S.; Hoffmeister, M.; Zimermann, P. The role of the Industry 4.0 asset administration shell and the digital twin during the life cycle of a plant. In Proceedings of the 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Limassol, Cyprus, 12–15 September 2017; Available online: https://ieeexplore.ieee.org/abstract/document/8247583/?casa_token=jrAJiR2mMhYAAAAA:2r9j8V6_qTY63f3Cl_b2H3ugFV0s_EhzHsTMYljZNNYiDoTndYXIIdskBPT_uP_wJf5kkJE (accessed on 9 March 2023).

54. Bradac, Z.; Marcon, P.; Zezulka, F.; Arm, J.; Benesl, T. Digital twin and AAS in the Industry 4.0 framework. *IOP Conf. Series: Mater. Sci. Eng.* **2019**, *618*, 012001. [CrossRef]
55. Park, K.T.; Son, Y.H.; Ko, S.W.; Noh, S.D. Digital twin and reinforcement learning-based resilient production control for micro smart factory. *Appl. Sci.* **2021**, *11*, 2977. [CrossRef]
56. Talkhestani, B.A.; Jung, T.; Lindemann, B.; Sahlab, N.; Jazdi, N.; Schloegl, W.; Weyrich, M. An architecture of an intelligent digital twin in a cyber-physical production system. *Automatisierungstechnik* **2019**, *67*, 762–782. [CrossRef]
57. Lehnert, C.; Engel, G.; Steininger, H.; Drath, R.; Greiner, T. A hierarchical domain-specific language for cyber-physical production systems integrating asset administration shells. In Proceedings of the 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vasteras, Sweden, 7–10 September 2021; pp. 1–4. [CrossRef]
58. Arm, J.; Benesl, T.; Marcon, P.; Bradac, Z.; Schröder, T.; Belyaev, A.; Werner, T.; Braun, V.; Kamensky, P.; Zezulka, F.; et al. Automated design and integration of asset administration shells in components of Industry 4.0. *Sensors* **2021**, *21*, 2004. [CrossRef] [PubMed]
59. Culot, G.; Fattori, F.; Podrecca, M.; Sartor, M. Addressing Industry 4.0 cybersecurity challenges. *IEEE Eng. Manag. Rev.* **2019**, *47*, 79–86. [CrossRef]
60. Kuhn, T.; Schnicke, F.; Antonino, P.O. Service-based architectures in production systems: Challenges, solutions & experiences. In Proceedings of the 2020 ITU Kaleidoscope: Industry-Driven Digital Transformation (ITU K), Ha Noi, Vietnam, 7–11 December 2020. [CrossRef]
61. Javaid, U.; Aman, M.N.; Sikdar, B. A scalable protocol for driving trust management in internet of vehicles with blockchain. *IEEE Internet Things J.* **2020**, *7*, 11815–11829. [CrossRef]
62. Legat, C.; Vogel-Heuser, B. A configurable partial-order planning approach for field level operation strategies of PLC-based industry 4.0 automated manufacturing systems. *Eng. Appl. Artif. Intell.* **2017**, *66*, 128–144. [CrossRef]
63. Cavalieri, S.; Salafia, M.G. A Model for predictive maintenance based on asset administration shell. *Sensors* **2020**, *20*, 6028. [CrossRef] [PubMed]
64. Beden, S.; Cao, Q.; Beckmann, A. Semantic asset administration shells in industry 4.0: A survey. In Proceedings of the 2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS), Victoria, BC, Canada, 10–13 May 2021; IEEE: Piscatvey, NJ, USA, 2021; pp. 31–38.
65. Borgo, S.; Leitão, P. Foundations for a core ontology of manufacturing. In *Ontologies. Integrated Series in Information Systems*; Springer: Boston, MA, USA, 2007; pp. 751–775. [CrossRef]
66. Huang, Y.; Dhouib, S.; Medinacelli, L.P.; Malenfant, J. Enabling semantic interoperability of asset administration shells through an ontology-based modeling method. In Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, Montreal, QC, Canada, 23–28 October 2022; pp. 497–502.
67. Shukla, A.K.; Janmaijaya, M.; Abraham, A.; Muhuri, P.K. Engineering applications of artificial intelligence: A bibliometric analysis of 30 years (1988–2018). *Eng. Appl. Artif. Intell.* **2019**, *85*, 517–532. [CrossRef]
68. Vasilis, S.; Nikos, N.; Kosmas, A.; Dimitris, M. A toolbox of agents for scheduling the paint shop in bicycle industry. *Procedia CIRP* **2022**, *107*, 1156–1161. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.