IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 5 November 2025, accepted 30 November 2025, date of publication 4 December 2025,
date of current version 11 December 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3640265

RESEARCH ARTICLE

An Agent-Based Approach for Implementing Asset Administration Shell Type 3

LUCAS SAKURADA[1,2], FERNANDO DE LA PRIETA[3], AND PAULO LEITAO[1,2], (Senior Member, IEEE)

[1]Research Centre in Digitalization and Intelligent Robotics (CeDRI), Instituto Politécnico de Bragança, Campus de Santa Apolónia, 5300-253 Bragança, Portugal
[2]Laboratório para a Sustentabilidade e Tecnologia em Regiões de Montanha (SusTEC), Instituto Politécnico de Bragança, Campus de Santa Apolónia, 5300-253 Bragança, Portugal
[3]BISITE Digital Innovation Hub, Edificio Multiusos I+D+i, University of Salamanca, 37007 Salamanca, Spain

Corresponding author: Lucas Sakurada (lsakurada@ipb.pt)

This work was supported by FCT - Fundação para a Ciência e Tecnologia, I.P. by projects: CeDRI, UID/05757/2025 (DOI: 10.54499/UID/05757/2025) and UID/PRR/05757/2025 (DOI: 10.54499/UID/PRR/05757/2025); SusTEC, LA/P/0007/2020 (DOI: 10.54499/LA/ P/0007/2020). The work of Lucas Sakurada was supported by FCT under Grant 2020.09234.BD (DOI: 10.54499/2020.09234.BD).

ABSTRACT Industry 4.0 (I4.0) is driving the digitization of traditional manufacturing systems toward intelligent and adaptable factories, where industrial assets are interconnected and capable of collaborating inside and cross-company to enhance production processes. A key enabler of this transition is the Asset Administration Shell (AAS), a standardized digital representation of an asset that encapsulates all asset-relevant information throughout its lifecycle. The AAS ensures that all assets, regardless of their type or function, can be seamlessly integrated into the I4.0, facilitating interoperability and efficient data exchange across the entire production network. However, traditional AAS solutions, namely AAS Types 1 and 2, cannot address all the requirements of modern industry environments. To overcome these limitations, the AAS Type 3 (proactive) extends beyond the conventional functionalities of traditional AAS by incorporating more sophisticated features, enabling I4.0 components to operate with greater autonomy, intelligence, and collaborative capabilities. While traditional AAS implementations (Types 1 and 2) have been widely studied and adopted, research on AAS Type 3 is still in its early stages, lacking formal specifications and standardized guidelines for its design and implementation. In this context, this article presents an agent-based AAS approach for implementing AAS Type 3, offering readers a clearer, more accessible description of its features and practical applicability while encouraging further research and development in this emerging topic. For this purpose, several experiments were performed to gain a better understanding of its application in industrial scenarios. Additionally, the article discusses the experiences, insights, limitations and future perspectives derived from the design and experimental implementation of the agent-based AAS approach.

INDEX TERMS Asset administration shells, cyber-physical systems, industry 4.0, multi-agent systems.

I. INTRODUCTION

A. CONTEXT AND MOTIVATION

Traditional control approaches are based on centralized and hierarchical structures, typically following the well-known ISA-95 standard. Despite these approaches being effective for optimizing production processes, they often lack the agility and adaptability to respond quickly to the ever-changing market characterized by high-customized and high-quality products. However, with the emergence of Industry 4.0 (I4.0), a paradigm shift is taking place toward decentralized control architectures designed to overcome these limitations through the adoption of Cyber-physical Systems (CPS) [1]. CPS play an important role in this vision, facilitating the design of large-scale systems endowed with intelligent, highly automated, and adaptable functions based on a set of distributed and autonomous entities, which combine cyber and physical counterparts to interact and collaborate in order to reach the system's goals.

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

© 2025 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License.
For more information, see https://creativecommons.org/licenses/by/4.0/

A key challenge in implementing CPS is achieving seamless integration and communication between these cyber-physical components, which often leads to the development of ad-hoc solutions. Although these solutions may partially address the integration issues, they prevent establishing standardized, scalable, and interoperable systems, leading to compatibility issues and increased complexity, ultimately preventing the wide adoption and long-term sustainability of CPS-based systems. To address this challenge, Reference Architecture Model Industrie 4.0 (RAMI4.0) [2] is a three-dimensional model that formalizes all essential aspects related to the digitization of industrial assets, providing guidelines and a common understanding to the participants in developing I4.0-compliant solutions based on industrial standards. Central to RAMI4.0 is the concept of the I4.0 component, a specific specialization of CPS concept, encompassing an asset and its digital counterpart, the Asset Administration Shell (AAS) [3], [4].

The AAS, introduced in 2015 [5], [6], serves as a standardized digital representation of an asset, encapsulating all asset-relevant information throughout its lifecycle [3]. About a decade later, AAS has become a key enabler of interoperability within I4.0 environments, where seamless integration and communication between various systems, devices, and platforms are essential. Today, the Industrial Digital Twin Association (IDTA), a spin-off of Plattform Industrie 4.0, plays a key role in the standardization and development of AAS. According to recent IDTA specifications and research works in the literature, the focus remains on consolidating and refining AAS traditional solutions, namely AAS Type 1 and 2 solutions [4]. Although these solutions offer considerable advantages in standardizing asset representation and enabling interoperability, they need to be enhanced to handle dynamic and complex industrial scenarios that demand a higher level of autonomy and intelligence from I4.0 components for decision-making and collaboration.

Looking ahead, the future vision for AAS involves the development and standardization of AAS Type 3 (proactive), which extends beyond the conventional functionalities of traditional AAS by incorporating higher levels of autonomy, intelligence, and collaborative capabilities. These advanced features are indispensable for I4.0 components and play a critical role in realizing the complete vision of I4.0 of fully connected, intelligent, resilient, and adaptive industrial ecosystems. However, the progression toward AAS Type 3 is currently hindered by a lack of detailed specifications and guidelines for its implementation [4]. This gap presents significant challenges for developers and engineers in designing such components.

In this context, given their conceptual alignment, Multi-agent Systems (MAS) [7] are a suitable approach to support the realization of AAS Type 3 [8], providing the required autonomy, intelligence, and collaborative capabilities, which are attributes inherent to the software agents. MAS can be defined as a society of intelligent, autonomous, and cooperative agents that represent the physical or logical objects of a system. In such systems, the global behavior emerges from the interaction between individual agents through the implementation of interaction strategies, e.g., collaboration and negotiation. Additionally, agents can be equipped with learning mechanisms to dynamically adapt their state and behaviors in response to condition changes.

B. CONTRIBUTION

This article introduces a key dimension to the intelligence and collaborative capabilities of I4.0 components, presenting an approach that seamlessly integrates agents into the existing AAS structure with the objective of implementing the concept of AAS Type 3. In this regard, the contribution of this article is twofold: (i) to provide specifications and guidance for researchers and practitioners aiming to advance the design and implementation of AAS Type 3 through an agent-based approach, and (ii) to stimulate further exploration and development in this emerging field by offering a clearer, more accessible description and practical applicability of AAS Type 3. To this end, a series of experiments were conducted applying the proposed agent-based AAS approach across various industrial scenarios. Furthermore, the article reflects on the experiences and insights gained from the design and experimental implementation of the agent-based AAS approach, providing valuable input for future refinement and practical application.

C. SCOPE DELIMITATION

A recurring question in the I4.0 community concerns mainly the relationship between the AAS and the Digital Twin (DT), as both are defined as digital representations of assets. Several studies (e.g., [9], [10], [11], to name a few) have investigated this relationship, revealing different perspectives. For instance, some sources consider the AAS synonymous with or as a direct implementation pathway toward DTs, while others consider it as a foundational information model that supports interoperability among DTs.

In this work, the relationship between the AAS and the DT is interpreted in terms of their complementary roles. The DT represents a conceptual vision of a dynamic, data-driven, and intelligent digital counterpart of an asset, emphasizing synchronization, monitoring, control, and lifecycle integration. On the other hand, the AAS provides a standardized information model and interfaces that defines how such a digital representation can be structured, managed, and exchanged within the I4.0 ecosystem.

In this context, AAS Type 3 embodies the convergence of these two concepts. By extending the AAS with autonomy, intelligence, and collaborative capabilities, it evolves from a descriptive data container into a proactive, self-managed digital entity capable of decision-making and collaboration with other assets. Therefore, AAS Type 3 can be regarded as a practical implementation layer of the DT concept [11], ensuring that digital representations are not only standardized

and interoperable but also adaptive and context-aware in their interactions.

Nevertheless, it is important to emphasize that the primary objective of this work is to support the practical realization of AAS Type 3 within the RAMI4.0 context and in accordance with the original definitions and purposes established by Plattform Industrie 4.0 and the IDTA, addressing the current lack of formal specifications and implementation guidelines. In this regard, the full implementation of the broader set of requirements typically associated with DTs, such as those outlined in standards like ISO 23247 [12], [13] and in the Digital Twin Capabilities Periodic Table [14] proposed by the Digital Twin Consortium, are beyond the scope of this article and are considered directions for future work. Therefore, the scope of this work is specifically centered on an agent-based approach that enables key features such as autonomy, intelligence, and collaboration within the AAS framework, thereby advancing the conceptual and practical realization of AAS Type 3 in alignment with the I4.0.

Moreover, this work aligns with the trend toward modularization of production resources, as introduced in the Capability–Skill–Service (CSS) model [15], where the description of functionalities and the definition of standardized interfaces to encapsulated automation functions are key elements. This modularization principle conceptually supports the representation of *capabilities* and *skills* later discussed in this article.

D. ARTICLE ORGANIZATION

The rest of the article is organized as follows: Section II overviews the AAS concept and discusses some research works related to the AAS Type 3 concept. Section III introduces the proposed agent-based AAS approach. Section IV outlines the modeling of the asset information to support the decision-making process and collaborative tasks of the agent-based AAS. Section V describes some examples of interaction patterns among these components. Section VI describes the case study for testing the agent-based AAS approach and Section VII presents the experimental implementation. Section VIII discusses the experiences, limitations and future perspectives based on the agent-based AAS approach's design and experimental implementation. Finally, Section IX concludes the article.

II. ASSET ADMINISTRATION SHELL OVERVIEW

A. DEFINITION, STRUCTURE AND TYPES

The AAS is defined as a standardized digital representation of an asset that manages and encapsulates all asset-relevant information throughout its lifecycle [3]. Once an asset is equipped with its own AAS, it becomes identifiable and manageable within the I4.0 ecosystem through standardized interfaces. This transformation promotes interoperability and enables new forms of collaboration that were previously difficult or even impossible to achieve.

The AAS has a modular structure composed of a collection of submodels, each representing specific information or functionalities of the asset, such as identification, capabilities, technical data, and operational data [3]. Additionally, depending on the context and system requirements, the AAS can be implemented into three types, namely passive (Type 1), reactive (Type 2), or proactive (Type 3) [16]. This classification, introduced by Plattform Industrie 4.0 and the IDTA, defines AAS capabilities in terms of interaction patterns for data exchange and the degree of autonomy in decision-making. Although slight variations in interpretation exist in the literature, the AAS community generally adopts the following definitions [8], [16]:

- Type 1: represents the simplest type of AAS (file-based AAS) that stores the asset information throughout its lifecycle and can be exchanged as a whole in the form of a machine- and human-readable file, e.g., XML, JSON, and AASX.
- Type 2: functions as an API, responding to external requests but unable to make decisions or take initiatives. It allows for online access to asset information in a technology-neutral way.
- Type 3: represents a more advanced form, capable of making decisions and autonomously interacting with other AASs to exchange information and collaborate.

In contrast to Types 1 and 2, which mainly focus on information representation and standardized data exchange, Type 3 introduces a significant higher level of complexity by requiring the integration of intelligence, autonomy, and collaborative capabilities within the traditional AAS structure. These aspects pose considerable challenges not only in terms of coordination, robustness, scalability, and interoperability, but also regarding how such features should be conceptually designed and implemented. While Types 1 and 2 are relatively well defined, widely understood, and supported by official specifications (e.g., [17], [18]), the Type 3 remains in an early stage of development, with no formal guidelines currently available to support its realization. Consequently, its design and implementation details are still under discussion within the community. In this regard, the design of AAS Type 3 itself emerges as the key challenge, motivating the design-oriented approach proposed in this work to provide guidance for its realization.

As this work aligns with the concept of AAS Type 3, it is important to clearly define the minimum characteristics that such AAS must exhibit. Although official documentation (e.g., from Plattform Industrie 4.0 and the IDTA) does not outline these characteristics, insights derived from these documents and related research suggest that an AAS Type 3 should exhibit active behavior, including the ability to autonomously initiate communication, engage in negotiation, and make decisions [8]. This interpretation allows for a broad spectrum of possible characteristics for AAS Type 3. Accordingly, this work considers the following as essential characteristics for an AAS to be classified as Type 3 [7], [8], [19]:

- Autonomy: refers to the capability of a computational entity to operate without external intervention, maintaining control over its internal state and decision-making processes. In this work, autonomy does not mean absolute freedom to decide arbitrarily but rather the capability to make decisions based on predefined criteria.
- Intelligence: denotes the ability to perceive, reason, and act upon the environment to achieve predefined objectives. In this work, intelligence can have different levels, e.g., from basic rule-based mechanisms to advanced AI techniques, as well as collective intelligence.
- Collaboration: represents the coordinated interaction among multiple entities that share information, align goals, and cooperate to accomplish tasks that cannot be effectively achieved in isolation. In this work, collaboration refers to AASs working together for a particular purpose, extending beyond information exchange to a deeper, interdependent process where each participant contributes to achieving a common goal.

It is important to note that these are the expected minimum characteristics to be obtained by implementing an AAS Type 3 solution. However, this does not exclude the fact that the implemented solution can present other characteristics, e.g., adaptability, robustness, and pluggability, which are related to and/or resulted from these minimum characteristics.

B. RELATED WORK

This section provides an overview of research related to AAS Type 3 based on a literature search conducted in the Scopus database. The criteria for identifying if a paper relates to an AAS Type 3 solution is whether the proposed approach demonstrates some aspect related to intelligence, autonomy, or collaborative capabilities, as previously described. The classification criteria distinguish between complete AAS Type 3 solutions, which satisfy all characteristics, and partial solutions, which address only one or two aspects. Partial solutions were also considered, as they offer valuable insights that can help in the implementation of an AAS Type 3. Table 1 categorizes the main research works identified, followed by a discussion of some of these studies.

TABLE 1. Summary of the main research works related to the AAS Type 3.

Research works	AAS Type 3 characteristics		
	Autonomy	Intelligence	Collaboration
[20]–[23]	–	✓	–
[24]–[29]	–	–	✓
[30]–[33]	–	✓	✓
[8], [34]–[41]	✓	✓	✓

A predictive maintenance model based on AAS is proposed in [20], which mainly involves a data flow between AASs at the Edge and advanced logical blocks based on Machine Learning (ML) in the Cloud. The asset data is analyzed in the Cloud, where decisions are made and transmitted to the AASs. Despite the solution introducing some level of intelligence to the system, each AAS lacks the autonomy to make decisions on its own, as decisions are exclusively made in the Cloud. Additionally, there is no collaboration between the AASs, as only pre-designed data flows for data collection and aggregation are present. In line with this perspective, [23] proposes an architecture where the AAS facilitates interoperability with external predictive maintenance services and applications, contributing to increase the system intelligence.

An innovative approach that integrates Large Language Models (LLMs) with AASs is proposed in [21]. This work uses LLM to interpret complex information and support decision-making processes in industrial automation systems. The information is sourced from AASs, which provide descriptive details about a specific asset. Although LLM can enhance the decision-making process, the authors highlight several practical challenges that should be considered before adopting this type of solution. Furthermore, aspects related to autonomy and collaboration are not addressed in detail.

In [22], a systematic method for generating and deploying data analytics models in AASs is presented. This method focuses on generating energy prediction models using AI techniques and converting these models into standardized formats. After conversion, these models are integrated into AAS submodels, which external clients can request and utilize for energy prediction. While this method significantly enhances interoperability in manufacturing intelligence, it does not address the autonomy and collaborative aspects.

The authors in [26] expands the factory boundaries by proposing an AAS-based solution to enable shared production scenarios. In this scenario, trusted partners can dynamically share manufacturing capabilities and services over digital interfaces throughout the product lifecycle. The process involves a negotiation to find suitable manufacturers for customized products. This negotiation is facilitated by a platform, so the interaction is not directly between AAS entities. The paper focuses more on modeling AAS submodels and presenting examples of interaction patterns (related to collaboration) using the I4.0 language [42], [43] rather than providing details about practical implementation and adopted technologies. Also, [28] illustrates an I4.0-compliant digitization of a laser cutter module using AAS, where communication with the AAS is facilitated through the use of I4.0 language via MQTT.

The authors in [30], [31], [32], and [33] propose solutions using OPC UA to enable communication between AASs. However, their approaches separate the intelligence and decision-making process from the AAS. For instance, in [33], a higher management layer focuses on service-oriented functions such as monitoring, scheduling, control, and anomaly detection. This upper layer retrieves asset information to monitor and control the process and make decisions. While the overall system can be considered intelligent, the AAS itself is not an autonomous and intelligent entity.

Among the research works that cover all the characteristics, those based on MAS stand out. For example, [36] proposes design patterns for implementing AAS using industrial agents, and [37] presents an agent-based approach to enhance the digitization process of industrial assets, enabling the development of intelligent and collaborative AASs. Other studies focus on using the AAS to enhance MAS solutions. For instance, [34] discusses how asset information described in the AAS can serve as a standardized knowledge representation for agents, allowing them to access all relevant information for decision-making from the AAS submodels. Similarly, some research works focuses on modeling AAS submodels for agents to be used in different applications, namely dynamic replanning [35], shared production [39], scheduling problems [40], and fault detection and diagnosis [41]. Additionally, [38] proposes an approach that integrates MAS with AAS to enable a human-centered production paradigm, extending the traditional collaboration between AAS for products and resources by incorporating AAS representing human operators into the process. Finally, [8] investigates agent-based AAS approaches in more detail, highlighting the role of MAS in realizing AAS Type 3. However, it lacks concrete specifications and empirical validation, remaining a purely conceptual and theoretical study that serves as groundwork for subsequent research.

Based on this review, it was identified that while many research works make significant contributions, most do not address all the characteristics necessary for developing complete AAS Type 3 solutions. On the other hand, research works based on MAS fully address these aspects. Although existing research provides both theoretical and practical evidence supporting the use of MAS to implement AAS Type 3, the preliminary or conceptual nature of most of these studies and their tailoring to specific applications limit their broader applicability. For such approaches to gain wider adoption, they must be elevated to a higher level of abstraction that enables generalization beyond their original contexts. In this regard, and considering the early-stage nature of current research along with the lack of guiding specifications, the following section introduces an agent-based approach to support the design and implementation of AAS Type 3 solutions, capable of supporting a wide range of collaboration strategies and applicable to different industrial scenarios.

III. AGENT-BASED AAS APPROACH

This section presents an agent-based AAS approach that focuses on seamlessly integrating agents into the existing AAS structure. The goal is to implement the concept of AAS Type 3, which emphasizes a higher level of autonomy, intelligence, and collaborative capabilities for I4.0 components.

A. OVERVIEW

As illustrated in Figure 1, in the proposed approach, MAS offer a well-suited architecture for establishing decentralized and highly adaptable autonomous systems that arise from collaborative interactions among multiple autonomous and cooperative agents. These agents may implement novel functionalities for the I4.0 components, encapsulating data analysis and AI algorithms to support monitoring, diagnosis, prediction, and optimization tasks. At the same time, AAS plays a key role in providing a standardized asset information model, which serves as a basis for achieving interoperability and facilitating seamless data exchange in the I4.0 ecosystem, as well as serving as an information source for agents. Based on that, agents can obtain the required asset-specific information to support in the decision-making process and collaborative tasks. Moreover, the decoupling of the proposed approach allows reusing implemented AAS Type 1 and 2 solutions, extending it to AAS Type 3 solutions through the integration with agents.

Additionally, the agent-based AAS approach enables both vertical and horizontal collaboration. Vertical collaboration is performed by components of different factory levels, e.g., shop floor and business levels. In this context, higher-level systems, e.g., MES and ERP, are able to exchange information with shop-floor devices, such as products and resources (e.g., machines and production units), to plan and optimize production. On the other hand, horizontal collaboration is performed by components at the same factory level, typically between products and resources, which are able to exchange production-related information and autonomously adapt to dynamic production requirements. The versatility of horizontal and vertical collaboration allows the system to move between hierarchical and heterarchical configurations depending on the requirements and conditions of the system. For example, higher-level systems can optimize production orders and allocate tasks to shop-floor devices, while shop-floor devices can have the intelligence and autonomy to collaborate with each other and make decisions to respond quickly to unexpected events, e.g., downtimes due to machine failure.

B. INFORMATION METAMODEL

From a more technical point of view, Figure 2 illustrates the available elements for modeling agent-based AAS metamodel instances. The proposed metamodel was developed based on a high-level reference model presented in [8]. While the reference model offers a general conceptual framework, the metamodel aims to provide a more specific and detailed representation of the required elements and their relationships for the practical implementation of the proposed approach, translating the high-level principles into concrete structures applicable to the context of AAS Type 3.

The metamodel consists of two major parts, namely information and intelligent part. The information part presents the main classes to describe the asset information in the AAS submodels. The intelligent part presents the main classes to enhance the AAS with autonomy, intelligence, and collaborative capabilities provided by the agent. Regarding the information part, it will not be covered in this document since it follows the specifications defined in [17]. However,

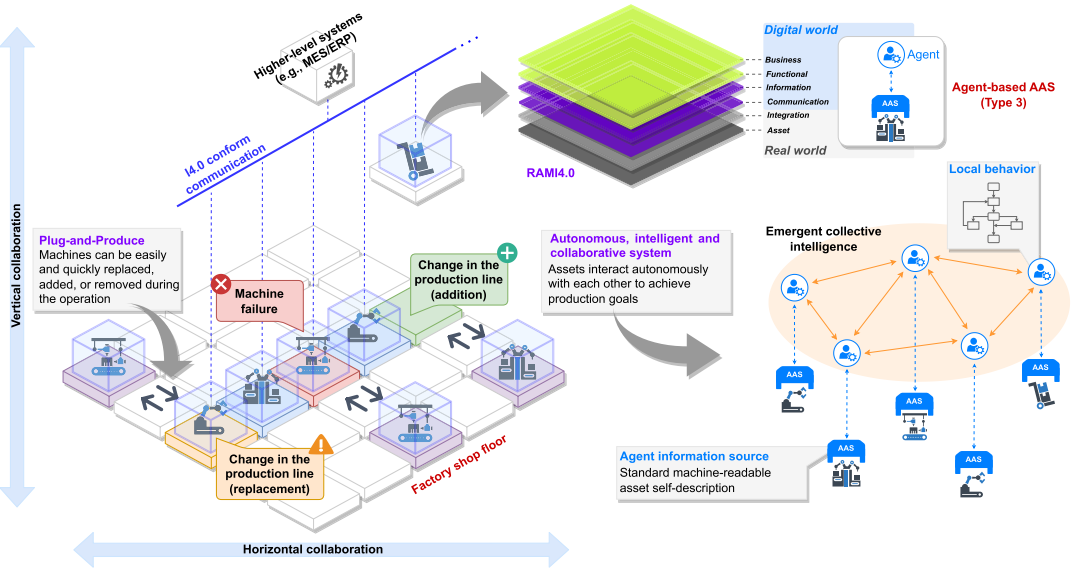


FIGURE 1. Overview of the agent-based AAS approach.

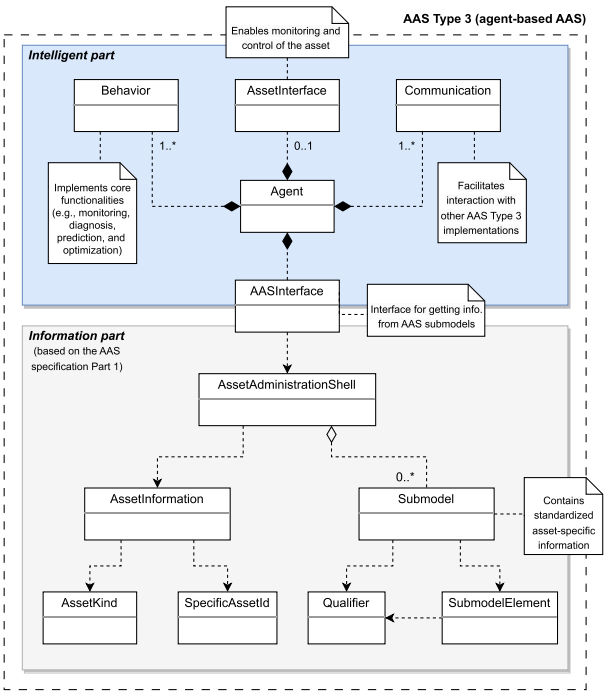


FIGURE 2. Agent-based AAS metamodel overview.

in summary, the *AssetAdministrationShell* is mainly composed of *Submodels* that describe the content or functional aspects of an asset in a standard manner. On the other hand, the intelligent part describes four classes that constitute an *Agent*, namely *Behavior*, *Communication*, *AASInterface*, and *AssetInterface*.

- The *Behavior* defines how the agent acts and makes decisions in various situations to achieve specific goals. Moreover, the behavior of an agent is fundamental as it encapsulates the intelligence and essential functionalities of the agent, such as monitoring, diagnosis, prediction, and optimization.
- The *Communication* encompasses the rules and interaction protocols that agents adhere to when interacting and collaborating with other agents, components, or systems, defining how agents exchange information, coordinate their actions, and participate in cooperative tasks.
- The *AASInterface* represents the agent's ability to access and retrieve information from the submodels, which it can then use to make decisions or to construct or enhance its knowledge representation. In this sense, the agent can better understand the system it operates and make more informed decisions.
- The *AssetInterface* allows the agent to interact with the asset, defining how the agent can request functions, access data, or set parameters on this asset. However, this interface is optional and only needed if the agent represents an asset that performs real-world functions. For example, if the agent represents a product that does not perform skills like drilling or assembling, the agent does not need to interface with the asset, e.g., to request those skills.

Based on this metamodel, the structure of an agent-based AAS can be set up in different ways. For example, the intelligent and information parts of an agent-based AAS can be decoupled and deployed along the Edge-Cloud layers, or if the asset has computational capabilities, the intelligent and information parts can be embedded directly within the asset. Independent of where the different parts are deployed, it is fundamental to define the proper interfaces that enable the interaction between the parts and with the asset.

IV. MODELING THE INFORMATION PART

As previously discussed, the information part of the agent-based AAS holds all information about the asset organized in a structured manner using submodels. Nevertheless, not every submodel has value or relevant information for the decision-making process, being necessary to define the proper submodels to help in the dynamic behavior of the agent-based AAS, particularly to support collaborative tasks. For this purpose, this section aims to describe fundamental submodels that enable the agent-based AAS to effectively manage and interact with its corresponding asset, as well as to enable collaboration between agent-based AASs. In summary, these submodels focus on providing the following information:

- Description of one or more functions performed by the asset, e.g., drilling and laser cutting.
- Description of how to interact with the asset, e.g., to execute asset services and adjust asset parameters.
- Description of the required operations for the manufacturing process of an asset and its manufacturing plan.

These submodels are mainly based on the concepts of capabilities and skills from the capability- and skill-based engineering domain and formally specified in the CSS reference model [15]. Moreover, this work employs official submodel templates provided by the IDTA to ensure greater interoperability. Nevertheless, there are some submodels that are currently under development or analysis for publication. Therefore, in specific cases, this work develops submodels that adhere to the AAS specification as needed.

A. CAPABILITIES: DESCRIPTION OF AUTOMATION FUNCTIONS

The CSS model defines capability as an abstract description of a function/operation performed by an asset with effects in the real world, e.g., drilling and laser cutting. Designing manufacturing systems based on the required capability for each step of the production process offers more agility and adaptability to the system. For example, this approach enables easier and quicker adaptation in scenarios like production replanning due to machine breakdowns, where having detailed capability descriptions can simplify the task of finding alternative machines with similar capabilities. On the other hand, in lot-size-one scenarios, the capability description allows for the identification of whether it is possible to manufacture a new product by evaluating which assets are capable of offering the required capabilities. In this context, it is fundamental to describe capabilities in a standard manner.

Currently, there is a submodel under development to express the capability of a production resource [44]. Since this submodel has not yet been officially released, this work considers modeling asset capabilities based on the submodel developed in [28], illustrated in Figure 3, which can be composed of a set of capabilities. For each capability, a specific submodel is instantiated and referenced, which consists of two *SubmodelElementCollection* classes, namely *TechnicalProperties* and *CommercialProperties*. These classes are customized with different properties according to the capabilities of each asset. *TechnicalProperties* refer to the description of the offered capability, e.g., a drilling machine can present the following properties: drilling diameter and depth. *CommercialProperties* refer to cost-related parameters, such as venue of provision and price. In this context, ECLASS [45] and IEC CDD [46], two dictionaries of standardized semantics based on the IEC 61360 standard, are recommended to describe these properties in an unambiguous, machine-readable, and industry-independent manner.

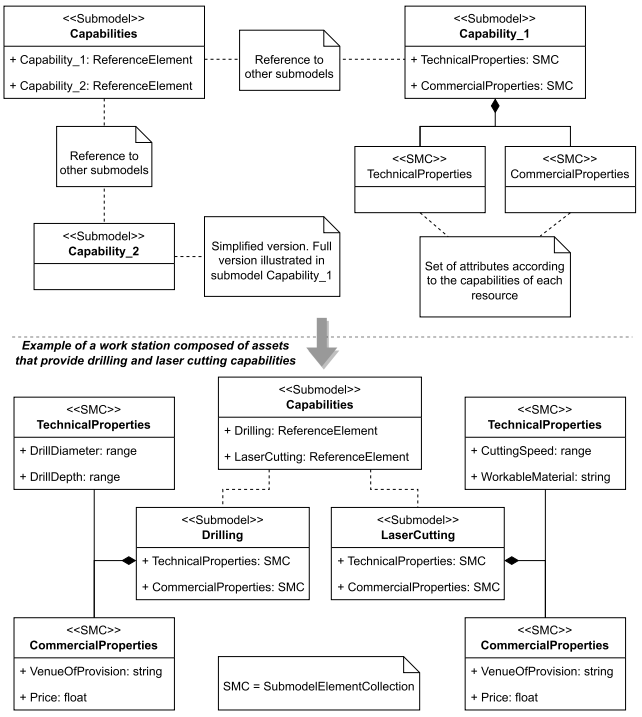


FIGURE 3. Capabilities submodel template.

B. SKILLS: INTERACTION WITH AUTOMATION FUNCTIONS

In manufacturing systems, physical assets are often connected to low-level controllers, such as programmable logic controllers, which follow the IEC 61131-3 and IEC 61499 standards to ensure real-time responsiveness and effective execution of the skills of their connected assets. According to the CSS model, a skill is the implementation of an encapsulated function/operation provided by a specific asset, i.e., a skill enables the realization of a capability. In this context, to facilitate the interaction with these assets, their skills should be exposed as services through standardized interfaces. In this sense, the description of the asset services or asset-related services should adhere to a standard format. This standardization is fundamental for ensuring seamless communication and integration across different components or systems in the manufacturing environment.

Based on that, IDTA provides standardized submodels that describe interaction patterns to facilitate the utilization

of skills through standardized interfaces, namely *ControlComponentType* [47], *ControlComponentInstance* [48], and *AssetInterfacesDescription* [49]. The first two are specific to the type of asset, whether it is a type or instance. Since this work considers assets as instances, the *ControlComponentInstance* submodel, complemented with the *AssetInterfacesDescription* submodel, is considered to model the asset skills.

The *ControlComponentInstance* submodel comprises two main *SubmodelElementCollection* classes, namely *Skills* and *Endpoints*. *Skills* refer to the collection of skills offered by the component, including information such as the parameters used to configure the skill, a reference to the error codes associated with the component that may be triggered by that skill, the designation of the operation in which the skill can be executed (e.g., manual, semi-automatic, and automatic), and references to other skills that are used by that particular skill. On the other hand, *Endpoints* serve as references to other submodels that describe how to interface with an asset based on communication protocols. In this sense, the *AssetInterfacesDescription* submodel can be used for this purpose since it specifies an information model for describing the interfaces of an asset service or asset-related services.

The *AssetInterfacesDescription* submodel is based on the Thing Description (TD) standard to define the content and structure of the submodel, describing in a standardized manner how to interface with assets that exhibit different protocols and payload formats, particularly using Modbus, HTTP, and MQTT. Since it is an extensive submodel with different schemas to support specific protocols, this work will not go into detail about all aspects of this submodel, describing only two fundamental *SubmodelElements*, namely *base* and *href* properties. The *base* property is the entry point for the asset connection, which is specified according to the protocol, e.g., the IP address and port of the asset. The *href* provides the description of asset endpoints. For instance, if the asset provides some service based on the HTTP protocol, the combination of *base* and *href* properties, e.g., `http://{address}:{port}/{endpoint}`, describes how to execute some service of the asset.

C. PRODUCT REQUIREMENTS AND MANUFACTURING PLAN

Based on the described submodels, a product should have a submodel to describe the required capabilities involved in its manufacturing process. In this context, it can be evaluated which assets are able to offer the required capabilities and skills, resulting in the product manufacturing plan. The product manufacturing plan can be described in a submodel structure, including the manufacturing steps, schedule, state, and the assigned resources to provide the capability in each step.

Since there are no official submodels related to product manufacturing, two submodels are designed to deal with these aspects. Figure 4 (left) illustrates the designed *ProductRequirements* submodel, which contains the required capabilities to manufacture a product. On the other hand, Figure 4 (right) shows the designed *ProductManufacturingPlan* submodel, which contains a *ListOfSteps* that describes one or more steps for producing a product. Each step results in a sub-product that, when its manufacturing plan is complete, becomes the final product. In this context, each *Step* describes information regarding the state, the scheduled start and end time, the required operation, the assigned resource to perform the operation, and a collection of requirements to perform the operation. The *Requirements* from both submodels depend on the type of product, so it is not possible to define a generic model since a product may require a drilling capability with the appropriate drilling requirements, such as diameter and depth, or a laser cutting capability with requirements such as cutting speed.

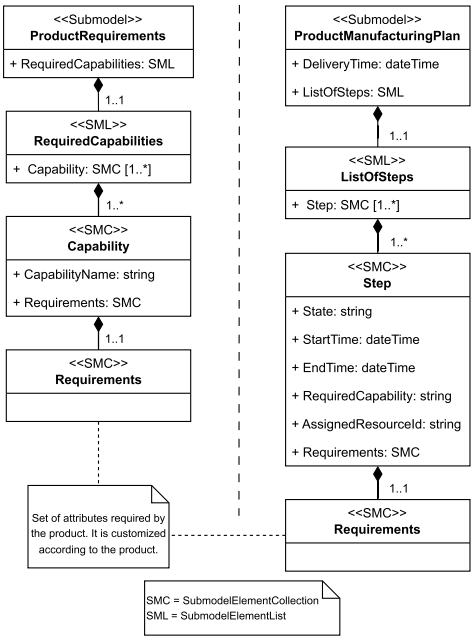


FIGURE 4. Submodel templates related to product manufacturing.

D. COMPLEMENTARY SUBMODELS

It is important to highlight that the proposed approach does not limit itself only to the previously described submodels. However, they provide the basis information to enable the collaborative tasks of agent-based AASs in manufacturing-related scenarios. Additional submodels can be introduced to describe specific functionalities performed by the agent-based AAS, such as monitoring, optimization, and diagnosis, which may rely on AI techniques. To support these functionalities, the adoption of AI-specific submodels, namely *AI Dataset* [50], *AI ModelNameplate* [51], and *AI Deployment* [52] could be adopted.

The main idea of the referred AI-related submodels is to cover the AI lifecycle in a standardized manner, ensuring that every stage, from dataset preparation to model deployment,

is systematically documented. Such standardization not only enhances the accessibility, reproducibility, and reusability of AI models but also provides greater transparency for stakeholders regarding the implemented functionalities and the decision-making criteria of agent-based AASs. By offering clear insights into how models are trained, deployed, and utilized, these submodels help build trust in AI-driven systems. While these submodels are not mandatory for supporting the collaborative tasks of an agent-based AAS, they can significantly enhance the system's capabilities by enabling more intelligent behaviors.

V. INTERACTION PATTERNS

The previous section focused on describing some fundamental submodels for the agent-based AAS, particularly providing the required information about the asset related to capabilities and skills, to support the management and interaction with the corresponding asset and the realization of collaborative tasks. Based on that, this section aims to present some examples of interaction patterns between them.

A. DISCUSSION ON FIPA AND VDI/VDE 2193 COMMUNICATION STANDARDS

Both FIPA and VDI/VDE 2193 (I4.0 Language) define interaction protocols that facilitate cooperation among autonomous entities, but they serve different purposes and levels of standardization. FIPA provides a well-established, domain-agnostic framework for agent communication, offering mature specifications and a comprehensive set of standardized interaction protocols that cover negotiation, coordination, and collaboration patterns. However, FIPA introduces additional overhead due to its Agent Communication Language (ACL) message structure, and its agent-oriented nature means it does not natively ensure interoperability within the I4.0 ecosystem.

On the other hand, VDI/VDE 2193 delivers a structured communication framework explicitly aligned with the I4.0 paradigm, bridging intelligent agent principles with real-world industrial communication. This ensures seamless integration with other I4.0 components, although the standard is still evolving and currently lacks the same level of maturity, tooling, and widespread adoption found in FIPA.

From an interoperability perspective, these two standards are complementary rather than competing. The interaction protocols defined by FIPA can be adopted and adapted within the VDI/VDE 2193 standard by introducing a mapping layer [8] between their respective message structures. Consequently, the proposed solution adopts VDI/VDE 2193 to ensure full I4.0 compliance, while reusing and adapting FIPA interaction protocols to enable coordination, negotiation, and decision-making among autonomous AAS entities. This approach effectively leverages well-established agent communication patterns within the AAS Type 3 context, combining the strengths of both standards.

B. EXAMPLE OF INTERACTION PATTERNS

Following the discussion on communication standards, this section exemplifies how interaction protocols can be adopted within the proposed agent-based AAS approach. The subsequent examples illustrate how these patterns support coordination, negotiation, and collaboration among autonomous AAS entities, forming the basis for collective behavior within the I4.0 environment.

As an example, Figure 5 shows the VDI/VDE 2193-2 Bidding protocol [43], which can support the autonomous negotiation processes between agent-based AASs in manufacturing environments, where they can act as service requesters and/or service providers. This protocol is particularly useful for scenarios where assets need to efficiently negotiate based on predefined rules and criteria to achieve the manufacturing goals. In this context, the service requester can represent a product that needs a certain service (e.g., the capabilities provided by a resource) to meet its requirements, and the service provider can represent a resource that offers its capabilities (e.g., drilling and milling) as a service at a certain cost. In this interaction pattern, the service requester initiates the interaction by performing a ''call for proposals'' containing a description of the required service and waits for responses from service providers. Service providers can then refuse or propose to provide the service by checking their feasibility. If they accept, they define an associated price and send their responses. Finally, the service requester evaluates the proposals and selects the best one based on decision-making mechanisms.

This protocol presents versatility to enable resource allocation based on vertical collaboration and system reconfiguration due to condition changes based on horizontal collaboration. In the first case, a higher-level system can negotiate with several shop-floor resources based on the capabilities they provide to determine which resources will be responsible for performing operations on the products to be manufactured. On the other hand, in the second case, if a machine designated to perform an operation breaks down, the product may be able to autonomously find and negotiate with other resources that are capable of performing the required capability.

Although the I4.0 Language is suitable for providing I4.0-compliant interactions, the current VDI/VDE 2193-2 standard specifies only the Bidding Protocol, which limits solutions that require a broader range of interaction patterns. In this context, and as previously discussed, the proposed agent-based AAS can also reuse the interaction protocols defined in the FIPA Agent Communication standards, adapting them when necessary to ensure I4.0 compliance. The main objective is to provide an adaptable and standardized communication framework that guarantees versatility for different collaborative scenarios.

For instance, Figure 6 illustrates the interaction pattern for the operation execution based on the FIPA Request Interaction protocol [53], where agent-based AASs can assume

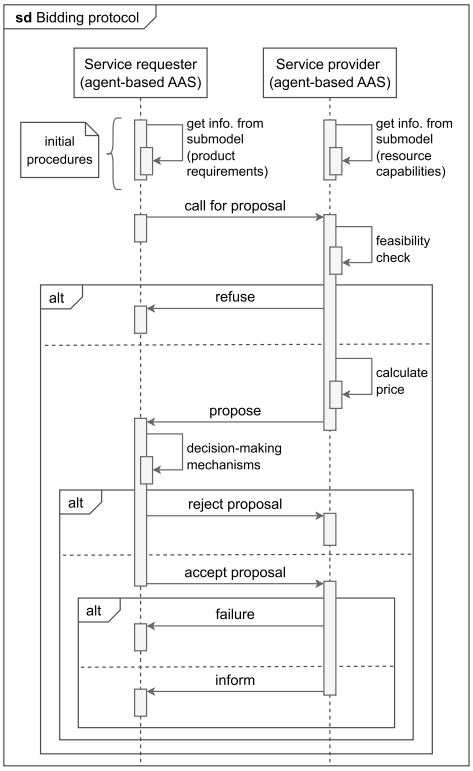


FIGURE 5. Interaction pattern for the autonomous negotiation process.

the role of initiators and/or participants. This interaction allows for a flexible and adaptive manufacturing process through the dynamic adjustment of process parameters based on real-time conditions and requirements. In such scenario, machines need to constantly adjust their operational parameters (skill parameters) to meet the production requirements. For instance, a product (initiator) can request the execution of an operation for a resource (participant). The product sends the parameters, and then the resource can decide whether to accept or refuse to perform the operation by checking the feasibility of the requested parameters. If accepted, the resource adjusts its operational parameters and performs the operation, informing the results at the end.

Another example is the interaction pattern for event notification based on the FIPA Subscribe Interaction protocol [54] to continuously receive information of interest or event notifications. For instance, if a machine breaks down, its corresponding agent-based AAS can publish a message notifying that the machine cannot perform the scheduled operations. In such scenario, all products designated for that machine are informed about the unexpected event and can attempt to redefine a new manufacturing plan based on the negotiations defined by the Bidding protocol (see Figure 5). In this context, agent-based AASs can assume the roles of subscribers and/or publishers, as shown in Figure 7. The subscriber needs to subscribe to a specific topic to receive the desired information, which can be periodic data updates or a warning when an event is triggered. On the other hand,

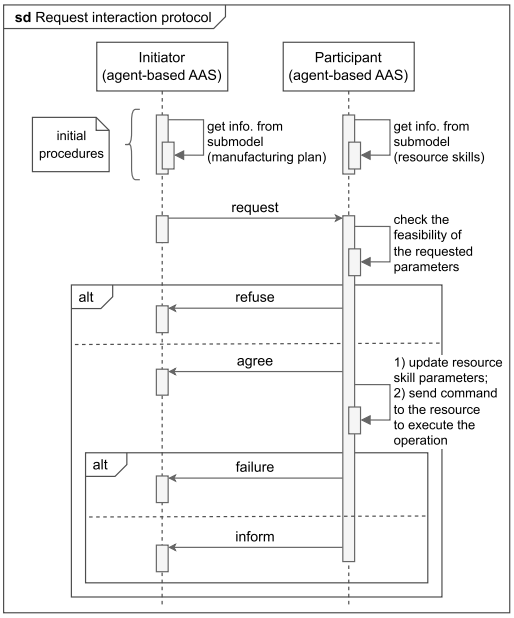


FIGURE 6. Interaction pattern for the operation execution.

the publisher will publish information on that topic when these circumstances are satisfied. Typically, this publish-subscribe model involves a broker, an intermediary entity that manages the exchange of messages between subscribers and publishers.

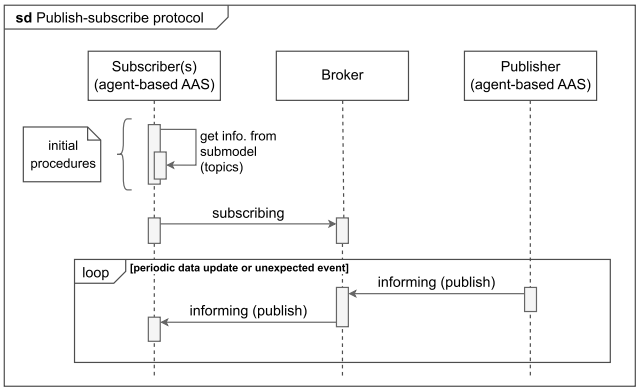


FIGURE 7. Interaction pattern for the event notification.

For this interaction pattern, a submodel containing a list of topics can be modeled, and the agent-based AAS can use this submodel to know which topics it needs to subscribe to in order to receive information. Based on the information received, the agent-based AAS can be designed to take the proper actions according to the specific use case. This protocol is particularly useful for broadcast messages, where a component needs to communicate with several other components. For instance, in a fault-tolerance scenario, an Autonomous and Mobile Robot (AMR) that experiences a locomotion failure but retains its communication capability

can publish a message to advise other AMRs not to use a congested route.

VI. DESCRIPTION OF THE CASE STUDY

To evaluate the collaborative aspects of the proposed solution, the agent-based AAS approach was designed, implemented, and experimentally tested in an extended emulated version of a small-scale production system demonstrator introduced in [55]. As previously discussed, the early-stage nature of research on AAS Type 3 makes a strict quantitative comparison with existing methods unfeasible at this point. Instead, this work focuses on the design-oriented realization of AAS Type 3, emphasizing the feasibility and effectiveness of its implementation principles rather than quantitative performance metrics. In this context, the demonstrator enables a qualitative assessment of how the proposed approach supports and enhances collaboration among multiple AASs, thereby verifying its conceptual compliance with the AAS Type 3 definition.

As shown in Figure 8 (bottom), the "real small-scale production system" comprises several assets, namely punching machines, indexed lines, and a transport robot provided by Fischertechnik. These assets realize different functions that emulate the operations of real machines on a small-scale model. For example, they can provide capabilities such as transporting, punching, and drilling workpieces. The low-level logic control of these machines is implemented as IEC 61131-3 programs running in a PLC. On the other hand, the transport robot executes its operations using programs developed in an ABB proprietary programming language that runs in the robot controller. Furthermore, the demonstrator has an API interface that enables the easy integration with the assets, such as checking a machine state, starting or stopping an operation, and configuring some conditions for specific tests, such as a machine breakdown, using MQTT or HTTP/REST protocols.

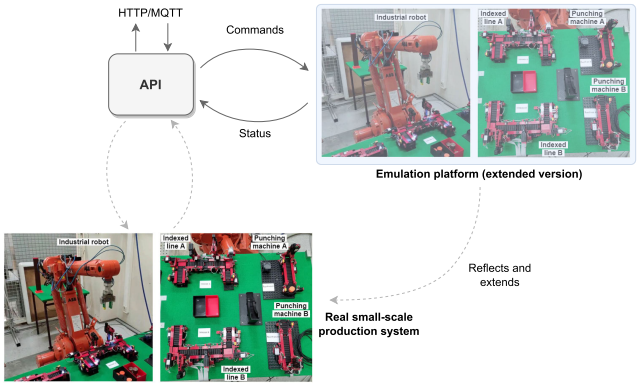


FIGURE 8. Real system and emulation platform for experimental tests.

Although the "real small-scale production system" enables several experimental tests, it is essential to increase the number of assets in the system to enable the creation of a more dynamic and complex environment that requires more interactions between the assets. In this sense, the experimental plan for this work is similar to the one adopted in [56]. The aim is to use an extended version of the current demonstrator setup through a developed emulation platform that abstracts real physical assets. This emulation platform enables replicating each asset and all its functionalities and interfaces (see Figure 8). From the API perspective, it is indistinguishable whether it is connected to the emulation platform or the real system. This approach allows to perform various experiments, such as trial and error tests, and scales the system regarding the quantity of assets in a flexible and safe virtual world. The extended version of the demonstrator includes three transport robots, three punching machines, three indexed lines, and n products.

For the experimental configuration, each emulated asset was assigned predefined operational parameters obtained from real measurements of the small-scale production system. The processing time for machines ranged between 17–26 s per operation, product transfers between stations required 12–16 s, and the system state was updated in the millisecond range. These parameters were selected to accurately reproduce the operational dynamics of the real system while maintaining a manageable simulation time frame.

It is important to note that the emulation platform operates entirely in software and therefore does not capture real-world latencies or hardware-level communication delays. In addition, message scheduling and synchronization rely on the frameworks used to implement the proposed approach, which introduces small variations. These aspects do not compromise the conceptual validity of the proposed approach but may influence precise timing analyses.

The extended small-scale production system can be easily configured to execute predefined production processes (e.g., initially processing a workpiece in the punching machine, followed by processing in the indexed line) using traditional control solutions capable of fulfilling specified tasks and achieving expected output volumes. However, when production is driven by high product customization and requires rapid adaptation to changing conditions, such conventional solutions become inefficient and costly to reconfigure. In this context, the agent-based AAS approach provides a promising alternative by decentralizing intelligence and enabling collaborative task execution among assets.

The goal of applying agent-based AASs to the demonstrator is to present the feasibility of the proposed approach in supporting collaborative behavior across different scenarios. These scenarios are derived from the Adaptable Factory (AF) scenario, which is one of the application scenarios [57] proposed by the Plattform Industrie 4.0 that describe a vision for the digital transformation of industrial systems toward the realization of I4.0. The AF scenario envisions a manufacturing environment that is highly flexible and responsive to changing demands, where machines, equipment, and products collaborate autonomously. Since the AF scenario is multifaceted, it can be broken down into several

sub-scenarios, namely *operation execution*, *autonomous negotiation*, and *unexpected event*. These scenarios address different aspects of the AF scenario that require a certain level of collaboration between assets to ensure efficient, flexible, and resilient production. Specifically, *operation execution* involves coordinating and executing tasks between various assets, namely products and resources, to fulfill production requirements. On the other hand, *autonomous negotiation* involves determining which machines and equipment should perform specific functions based on negotiation strategies and asset capabilities. Finally, *unexpected event* focuses on the system's ability to adapt and reconfigure in response to unexpected events, e.g., a machine breakdown.

VII. EXPERIMENTAL IMPLEMENTATION

This section presents the implementation of the proposed agent-based AAS approach and illustrates its application in realizing the outlined scenarios.

A. IMPLEMENTATION OF AN AGENT-BASED AAS

This subsection explains the general process of implementing an individual agent-based AAS using proper development platforms, which involves implementing the main classes described in the agent-based AAS metamodel.

1) INFORMATION PART

The AAS submodels were created using the Eclipse AASX Package Explorer tool V3.0 (https://github.com/eclipse-aaspe/package-explorer), a C#-based viewer and editor for developing AASs in accordance with the AAS information metamodel [17]. This tool allows AASs to be exported in several file formats, e.g., XML, JSON, and AASX. Therefore, it facilitates cross-company data exchange through file exchange methods and also provides the option to host these files on a server that supports a standardized API interface for accessing the information within these file-based AASs. Moreover, it offers an user-friendly graphical interface that makes it easy for non-programmer users to structure all asset information into submodels. Figure 9 illustrates the graphical interface of the Eclipse AASX Package Explorer tool, presenting some of the submodels instantiated for one of the indexed lines of the case study.

As previously described in Section IV, this work adopts official submodels provided by the IDTA, namely *ControlComponentInstance* [48] and *AssetInterfacesDescription* [49], as well as other defined submodels that comply with the AAS metamodel specification: *Capabilities*, *ProductManufacturingPlan* and *ProductRequirements*. The following sections will explain how agent-based AASs utilize these submodels in practice.

2) INTELLIGENT PART

The agents (intelligent part) were implemented using the Java Agent DEvelopment framework (JADE) (https://jade.tilab.com/), which implements an efficient agent platform and supports the development of MAS that complies with the FIPA specifications [53]. Regarding its features, JADE can be easily deployed on different computational platforms, providing advanced mechanisms to support the development of MAS for general applications, namely agent communication mechanisms, a library of FIPA interaction protocols ready to be used, a graphical user interface to manage several agents, and a set of useful tools to debug the developed agents.

JADE uses a platform model defined by FIPA that provides the infrastructure in which agents can be deployed, allowing the existence, operation, and management of agents. This model includes the Agent Management System (AMS), the Directory Facilitator (DF), and the Agent Communication Channel (ACC). The AMS is responsible for supervising the platform and managing the lifecycle of agents. The DF provides the yellow pages service, where agents can publish the services they provide and find other agents providing the services they need. The ACC manages communication between the agents.

Figure 10 illustrates the intelligent part (agent) of the agent-based AAS approach deployed in the JADE agent platform. In this context, JADE is responsible for providing all software infrastructure to manage the agents, while the information part (submodels) and the asset have their own management platforms.

The prototype was implemented using JADE version 4.5 in a distributed multi-container setup deployed on the same local network. The Main Container hosted the AMS and DF services, while multiple Peripheral Containers executed the agent-based AAS instances (one agent per asset). Communication relied on the ACC with ACL messages (default JADE serialization) over TCP/IP. Service discovery was achieved through DF registrations, where service types mirrored the corresponding AAS capabilities, and message tracing was conducted using JADE's Sniffer tool for debugging and monitoring purposes.

In this work, the agents (intelligent part) are Java classes that extend the `Agent` class provided by the JADE, inheriting basic functionalities such as registration services, remote management, and sending and receiving ACL messages. As previously described, an agent is comprised of four classes, namely *Behavior*, *Communication*, *AASInterface*, and *AssetInterface*. Each of these classes were implemented using the functionalities provided by JADE, namely the Behavior classes (e.g., CyclicBehaviour, OneShotBehaviour, TickerBehaviour, WakerBehaviour, and FSMBehaviour) and the library of ready-to-use FIPA interaction protocols (e.g., AchieveREInitiator, AchieveREResponder, ContractNetInitiator, and ContractNetResponder), as well as non-JADE libraries to support the use of communication protocols, e.g., MQTT (Eclipse Paho) and HTTP (Unirest).

Even though it is important to use the VDI/VDE 2193 [42], [43] standard to create communication structures in compliance with I4.0, this work used the FIPA ACL for message exchange instead, because JADE provides specific classes for using FIPA message structures. This option does not invalidate the solution, as it only reflects the structure

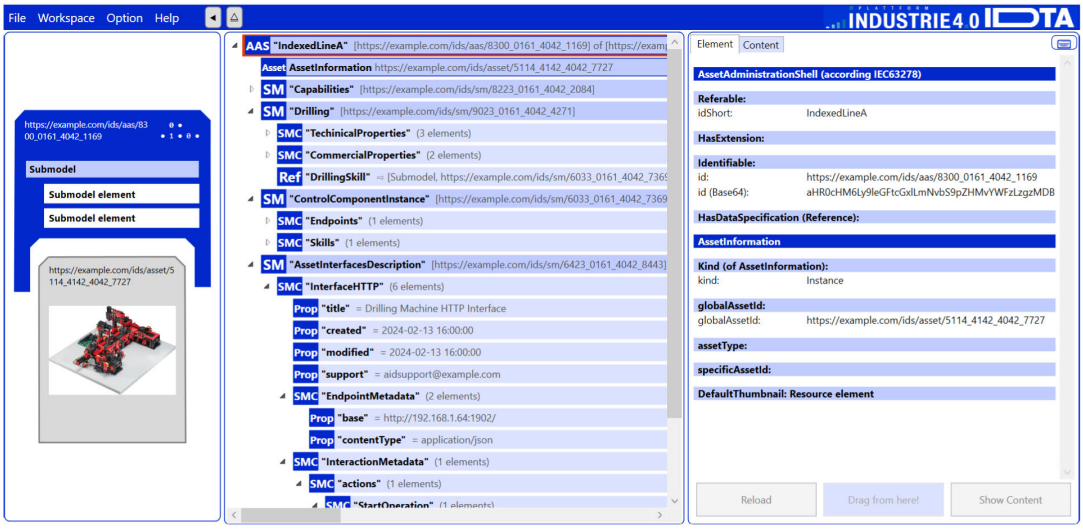


FIGURE 9. Developed submodels using Eclipse AASX Package Explorer tool.

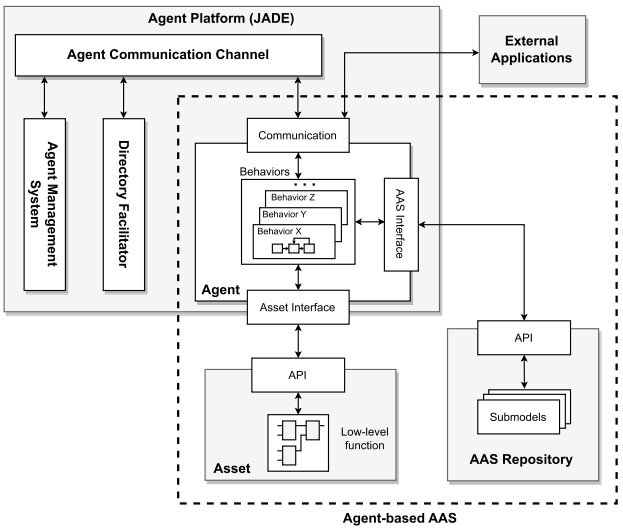


FIGURE 10. JADE agent platform to manage the intelligent part of an agent-based AAS.

and elements notation of the messages, not the collaboration and operational aspects of the proposed approach. Solutions requiring the VDI/VDE 2193-1 [42] message structure can easily adopt this structure.

It is important to note that JADE was adopted as a proof-of-concept platform to implement and validate the proposed agent-based AAS approach. Moreover, the proposed approach is technology-agnostic, meaning it is not dependent on JADE and can be implemented using other agent platforms or communication infrastructures as needed. The choice of JADE was motivated by its maturity, open-source availability, and comprehensive set of features for prototyping agent-based systems in compliance with FIPA, as previously discussed. These characteristics made it a suitable option for testing the proposed approach in a controlled and reproducible environment. Modern alternatives, such as Python-based MAS frameworks (e.g., Smart Python Agent Development Environment) or others reviewed in [58], may offer greater robustness, flexibility, scalability, or more seamless integration with AI techniques. Nonetheless, JADE fulfilled the requirements of this study.

3) API-ENABLED APPROACH

Eclipse AASX Server V3.0 (https://github.com/eclipse-aaspe/server?tab=readme-ov-file) complements the use of Eclipse AASX Package Explorer by providing a local service to host and serve AASX packages, enabling access to the asset information through the HTTP/REST, OPC UA, and MQTT protocols. The Eclipse AASX Server adheres to the AAS specification part 2 (version 3.0) [18], and therefore it can perform the operations described in [59]. This specification provides only guidelines for a HTTP/REST API, particularly describing the operations to enable access to the submodels information and their elements.

As shown in Figure 11, this API-enabled approach is crucial, since it provides standard interfaces for the agents to obtain information from their corresponding AASs using HTTP/REST. For example, the information of the submodels can be easily obtained through the GET method `/shells/{aasIdentifier}/submodels/{submodelIdentifier}`. This GET method example returns a specific submodel in a JSON object that needs to be interpreted via programming, i.e., by specific functions that extract information from this JSON object representing a particular submodel. Although these functions require customization to align with each submodel, the standardized submodel structure promotes code reusability, as a function developed to get information from a specific submodel can be reused by all agent-based AASs that have the same

submodel. Moreover, other methods are available, e.g., to edit or update the asset information using the POST, PUT, PATH, and DELETE methods.

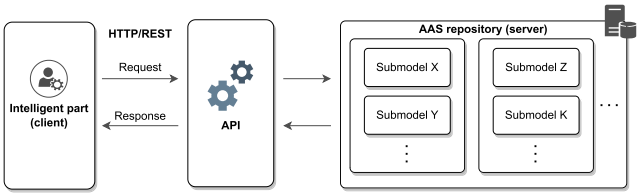
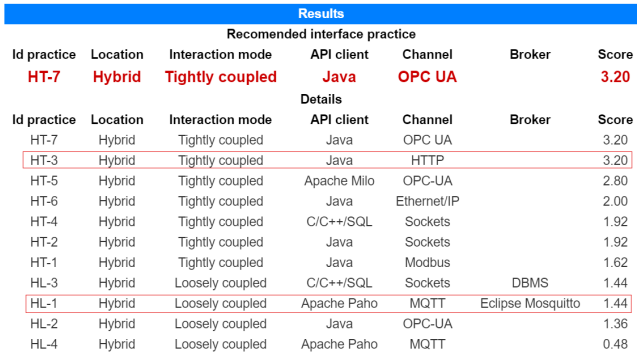


FIGURE 11. Integration between the intelligent and information part.



Results

Recomended interface practice

Id practice	Location	Interaction mode	API client	Channel	Broker	Score
HT-7	Hybrid	Tightly coupled	Java	OPC UA		3.20

Details

Id practice	Location	Interaction mode	API client	Channel	Broker	Score
HT-7	Hybrid	Tightly coupled	Java	OPC UA		3.20
HT-3	Hybrid	Tightly coupled	Java	HTTP		3.20
HT-5	Hybrid	Tightly coupled	Apache Milo	OPC-UA		2.80
HT-6	Hybrid	Tightly coupled	Java	Ethernet/IP		2.00
HT-4	Hybrid	Tightly coupled	C/C++/SQL	Sockets		1.92
HT-2	Hybrid	Tightly coupled	Java	Sockets		1.92
HT-1	Hybrid	Tightly coupled	Java	Modbus		1.62
HL-3	Hybrid	Loosely coupled	C/C++/SQL	Sockets	DBMS	1.44
HL-1	Hybrid	Loosely coupled	Apache Paho	MQTT	Eclipse Mosquitto	1.44
HL-2	Hybrid	Loosely coupled	Java	OPC-UA		1.36
HL-4	Hybrid	Loosely coupled	Apache Paho	MQTT		0.48

FIGURE 12. Ranking score of the recommended interface practices for the small-scale production system provided by the IEEE 2660.1 recommendation method tool.

4) ASSET INTEGRATION

Since the small-scale production system demonstrator offers two interface options, HTTP and MQTT, for interfacing with assets (see Figure 8), the IEEE 2660.1 recommendation method [60] can be used to determine the best of the two interface options. The IEEE 2660.1 recommendation method recommends the best interface practices to interconnect agents and physical assets, particularly low-level automation devices, based on user-selected criteria. These criteria include functions (e.g., control, monitoring, or simulation), application domains (e.g., factory automation, building automation, or energy and power systems), technological constraints (e.g., the ability to host agents in the asset controller), as well as the relevant characteristics for the application scenario (e.g., response time, scalability, and reusability). A key requirement for using the recommendation method is to have available scoring agent practices in a repository. This study involved a repository containing 23 interface practices that experts assessed (repository data provided by [60]).

In this context, using a developed tool that implements the IEEE 2660.1 recommendation method, the following criteria were defined: (i) the application domain is ''factory automation''; (ii) the asset cannot host agents locally; (iii) the functionality provided by the agent is the ''control'' of the asset, e.g., to start or stop an operation and adjust the production parameters that need to be changed according to the specifications of the product; (iv) the most important considerations in this configuration are ''response time'' and ''scalability'', each given a 50% weight. Based on these criteria, Figure 12 shows the results obtained from the tool. It indicates that the best practices for the selected criteria are the ''HT-7'' using OPC UA and ''HT-3'' using HTTP (score 3.2 out of 5). Despite the OPC UA interface (HT-7) being highly recommended for industrial environments, it was not selected because it is not available in the demonstrator setup. Additionally, the available MQTT interface (HL-1) was excluded due to its poor performance based on the defined criteria (score 1.44 out of 5). Therefore, this work adopts the interface using HTTP (HT-3).

Beyond the selection of the interface recommended by IEEE 2660.1, the proposed approach defines a modular and reusable mechanism to manage the interaction between the agent and the asset. This mechanism is supported through a dedicated submodel (e.g., *AssetInterfacesDescription*), which specifies the parameters, configuration data, and communication protocol required for asset integration. Based on the information provided in this submodel, the agent can dynamically instantiate and configure the appropriate interface module to communicate with the asset. Each module is designed to be protocol-specific (e.g., MQTT, HTTP, and OPC UA) but decoupled from the agent's internal control and decision logic, which remains independent of the underlying communication technology.

B. DEPLOYMENT OF THE PROPOSED APPROACH IN THE SMALL-SCALE PRODUCTION SYSTEM

This section presents the deployment of the agent-based AAS for the extended small-scale production system demonstrator described in Section VI. As shown in Figure 13, for each asset in the demonstrator, namely the industrial robot, indexed lines, punching machines, and products, an agent-based AAS was designed and implemented following the procedures previously described. It is important to note that the figure depicts the real demonstrator, not its extended version, serving only as an illustrative example to demonstrate the deployment of the proposed approach in the case study.

The testing process involves applying the agent-based AAS approach within the demonstrator to realize the previously described scenarios that require asset collaboration, namely *operation execution*, *autonomous negotiation*, and *unexpected event*. These scenarios should be understood as discrete conditions that may occur independently of each other. While they can occur sequentially or be interlinked based on production conditions, it is essential to recognize them as specific situations that can happen at certain moments within the manufacturing system that demand collaboration strategies for resolution. This perspective allows for a more problem-oriented evaluation of how the proposed approach can handle some collaborative production processes or some challenges that may arise within the manufacturing environment.

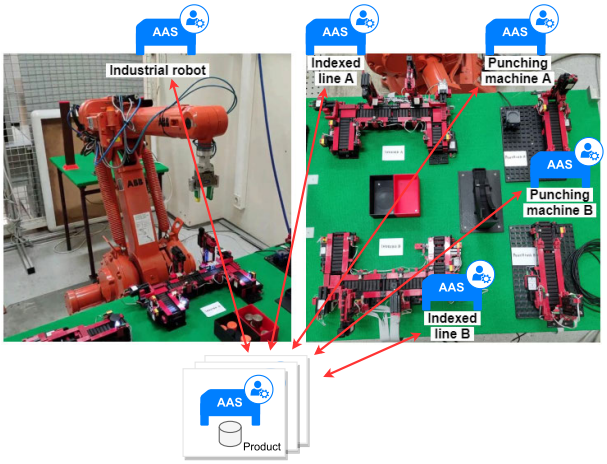


FIGURE 13. Conceptualization of the agent-based AAS approach applied in the demonstrator.

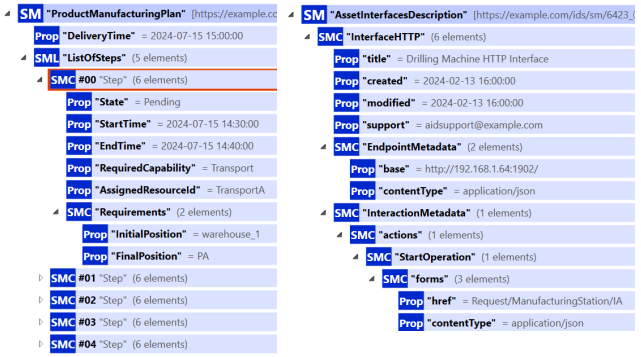


FIGURE 14. Short version of the developed *ProductManufacturingPlan* (left) and *AssetInterfacesDescription* (right) submodels.

1) OPERATION EXECUTION SCENARIO

This scenario focuses on the collaboration between products and resources to meet the production requirements following a predefined production plan. In this regard, the products must have information about their own manufacturing process, enabling them to navigate autonomously through various stages of production. As they advance through each stage, the products communicate with the corresponding resources to adjust process parameters based on their unique specifications. On the other hand, the resources must have the ability to manage its skills, e.g., to start or stop an operation, or adapt some parameters according to the product requirements. This interaction enables dynamic adjustments based on the specific needs of each product, enhancing flexibility in the production process. In this scenario, the defined production plan of a product comprises the following steps:

$$\textit{transport}\ (\textit{load}) \rightarrow \textit{punching machine} \rightarrow \textit{transport} \rightarrow \textit{indexed line} \rightarrow \textit{transport}\ (\textit{unload})$$

These steps are described and structured in the *ProductManufacturingPlan* submodel, as illustrated in Figure 14 (left). Among the properties of each step in *ListOfSteps*, *Requirements* provide the parameters to inform the resource that will perform the required operation. For example, the target positions are informed for the transport robot, and for the punching machine and indexed line, parameters related to the punching and drilling functions are informed.

Regarding the interface with the asset to manage its skills, Figure 14 (right) illustrates the *AssetInterfacesDescription* submodel, where information about how to interface with the asset is provided. In this case, the interface with an indexed line is performed via HTTP protocol, and one of the available services that starts the machine operation can be invoked through the endpoint `Request/ManufacturingStation/IA`.

Figure 15 illustrates the UML activity diagram describing the behavior of the agent-based AASs for the operation execution scenario. Figure 15 (left) shows the behavior of an agent-based AAS representing a product, which focuses on getting the information contained in the *ProductManufacturingPlan* submodel to know all processes involved in the product production. This submodel describes crucial aspects involved in each production process, such as the planned start and end times, assigned resource, state of the process, and requirements for the process. Based on this information, the agent-based AAS representing the product can interact with the agent-based AASs that represent the resources, requesting operations with the required parameters during the appropriate time slot. This behavior is cyclic, which means that it ideally ends when all planned processes are completed.

On the other hand, Figure 15 (right) shows the behavior of an agent-based AAS representing a resource. This behavior, in the first moment, aims to interface with the asset based on the description provided in the *AssetInterfacesDescription* submodel. After that, the agent-based AAS waits for requests to execute the requested operation.

Some conditions, such as failure or lack of connection with the asset, can prevent the production process. In these conditions, the agent-based AAS is designed to emit a warning message. However, for the purpose of this study, the experimental conditions in this scenario were structured to ensure that the production process was successfully completed without interruption. As a result, no specific actions have been implemented to handle these conditions. A particular condition, such as failure, is addressed in the unexpected event scenario.

As illustrated in Figure 6, the exchange of information between the agent-based AASs of the operation execution is based on the FIPA Request Interaction protocol [61]. In this interaction protocol, the products act as the "Initiator", requesting operations from the resources, which act as the "Participant". The resources receive the request along with the process parameters and then decide whether to accept or refuse to perform the operation by checking the feasibility of the requested parameters. If accepted, the resource adjusts

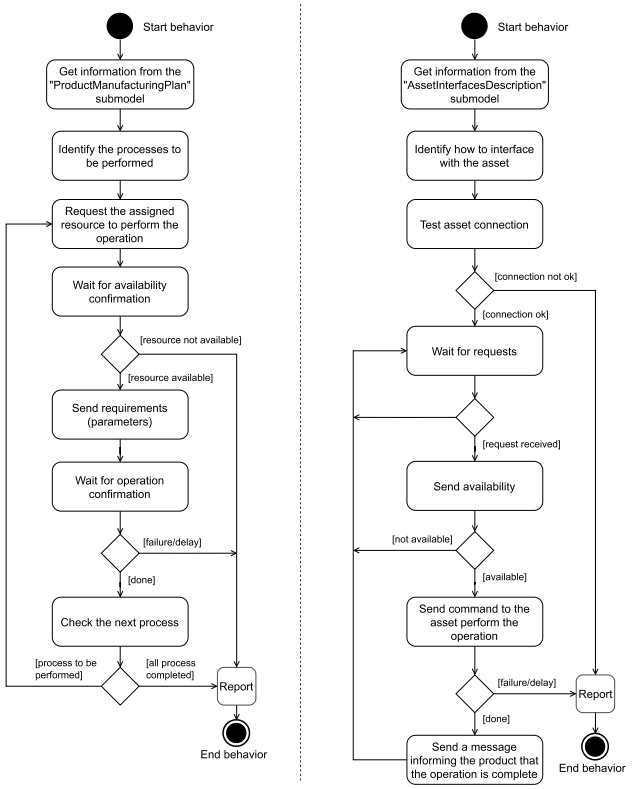


FIGURE 15. UML diagram of the operation execution behavior of an agent-based AAS representing a product (left) and a resource (right).

its operational parameters and carries out the operation, providing the results at the end.

2) AUTONOMOUS NEGOTIATION SCENARIO

This scenario focuses on the negotiation between products and resources in order to meet production requirements. In this context, products must know their own manufacturing requirements, and resources must know the capabilities they provide. This negotiation is based on a service-oriented principle, where the resources offer their capabilities as services, and the products can search for and request these services. In this scenario, products launched on the factory floor must autonomously negotiate with the available resources based on their required capabilities:

$$capability_1 \rightarrow capability_2 \rightarrow capability_3 \rightarrow \ldots \rightarrow capability_n$$

In this context, the products possess information about the required capabilities, described and structured in the *ProductRequirements* submodel, as shown in Figure 16 (left). For example, a product may require a drilling capability with specific requirements such as depth and diameter.

On the other hand, the resources have a collection of capabilities they can provide, described in the *Capabilities* submodel. Each element of this collection refers to a specific submodel that provides a detailed description of the capability. As shown in Figure 16 (right), the *Capabilities* submodel presents only one capability (drilling) that refers to

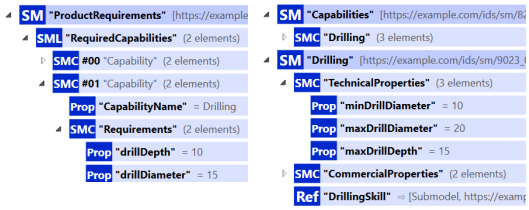


FIGURE 16. Short version of the developed *ProductRequirements* (left), *Capabilities* (right), and *Drilling* (right) submodels.

a specific *Drilling* submodel, which describes the technical and commercial properties of the drilling capability.

Figure 17 depicts the UML activity diagram that describes the behavior of the agent-based AASs in a negotiation scenario. Figure 17 (left) presents the behavior of the agent-based AAS representing a product, which focuses on acquiring information from the *ProductRequirements* submodel to understand the required capabilities for the product production process. In this sense, the product must engage in negotiations with the resources providing the required services, such as punching and drilling, as well as the transport between resources and warehouses (loading and unloading). The decision-making process for selecting the best proposal can be facilitated by decision-making algorithms, e.g., based on heuristic functions that consider the proposed price, the location of the resource, the proposed due date, and the confidence about the service provider (e.g., as proposed in [56]). In this work, the decision is based on choosing the cheapest price to facilitate the proof-of-concept.

On the other hand, Figure 17 (right) shows the behavior of the agent-based AAS representing a resource. The behavior begins with the agent acquiring information about the capability of its associated asset and registering this capability as a service using the Yellow Pages service implemented by JADE. The next step involves waiting for the call for proposals when products requiring the capabilities initiate negotiation by sending their production requirements. At this point, the agent-based AAS checks the availability of the resource and uses an algorithm to evaluate if the requirements match the technical specifications of the resource. This algorithm examines each requirement to ensure the resource can meet it. Based on this approach, the agent-based AAS representing the resource may choose to reject or continue the negotiation. If it agrees, it calculates the price and sends a proposal.

When calculating the price, various factors need to be considered, such as setup time and costs, expenses related to acquiring new tools, operational costs, resource investments, and the level of assigned work. This calculation process should be tailored to the specific interests of the company and is, therefore, customized for each case. Nevertheless, the dynamic bid price calculation presented in [56] is used as an example to illustrate this process in this work, which considers fixed costs and the profit margin adapted according to the market laws.

As shown in Figure 5, the exchange of information between the agent-based AASs of the negotiation process

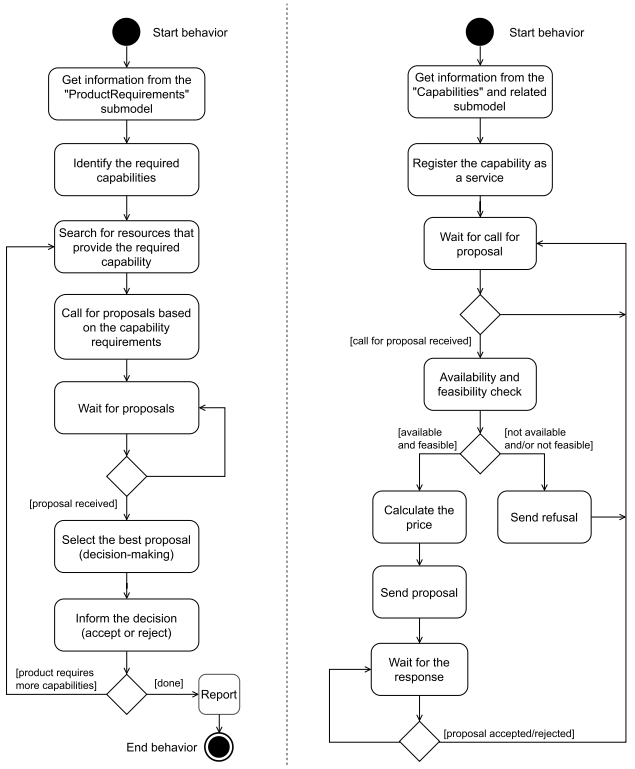


FIGURE 17. UML diagram of the negotiation behavior of an agent-based AAS representing a product (left) and a resource (right).

scenario is based on the VDI/VDE-2193-2 bidding protocol [43]. In this interaction protocol, the products act as the "Service requester", initiating the negotiation process with the resources acting as the "Service provider". When the resources receive the call for proposal, they can analyze and decide to continue the negotiation by offering proposals for the products. In this case, the products analyze the received proposals and decide on the best one.

3) UNEXPECTED EVENT SCENARIO

This scenario illustrates how a production system handles unexpected events, e.g., a machine malfunction that disrupts its normal operation. The focus is on the moment when the unexpected event occurs and how the agent-based AASs interact with one another to address this issue and ensure the completion of the production process.

Similar to the operation execution scenario, the system's initial state includes the requirement that products understand their manufacturing process to interact with various resources and that resources have the ability to manage their skills. Just like in the operation execution scenario, the production plan for the products consists of the following steps:

$$\textit{transport}\ (\textit{load}) \rightarrow \textit{punching machine} \rightarrow \textit{transport} \rightarrow \textit{indexed line} \rightarrow \textit{transport}\ (\textit{unload})$$

These steps are defined and organized in the *ProductManufacturingPlan* submodel, as depicted in Figure 14 (left). However, at a specific time, one of the assigned resources for carrying out one of these operations experiences a failure, necessitating the products to reschedule their production plans in order to complete the production process.

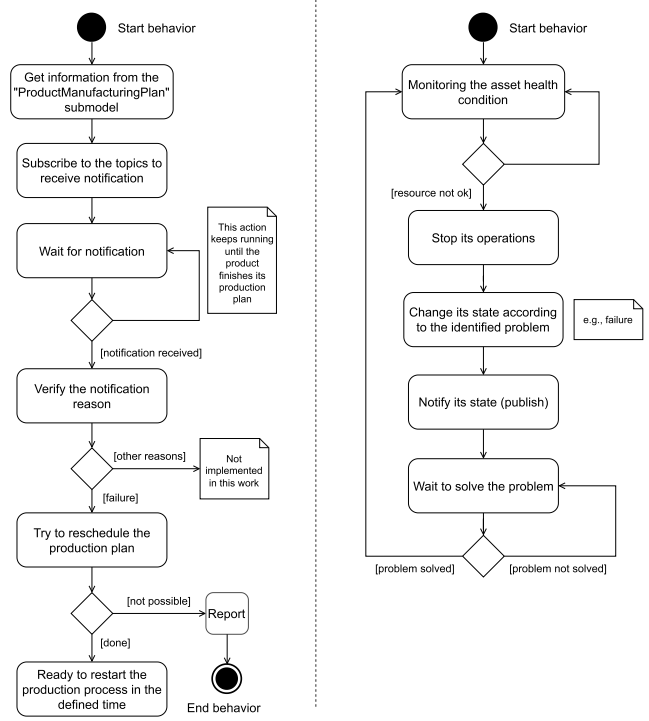


FIGURE 18. UML diagram of the unexpected event behavior of an agent-based AAS representing a product (left) and a resource (right).

Figure 18 depicts the UML activity diagram that describes the behavior of the agent-based AASs in an unexpected event scenario. Figure 18 (right) shows the monitoring behavior of an agent-based AAS representing a resource, which focuses on monitoring the health condition of its asset along its operation lifecycle. The behavior can utilizes various methods, from simple rule-based approaches to advanced AI algorithms, to monitor the asset. For instance, this work adopts an approach based on the Nelson rules [62] to determine whether any measured variable is out of control. By employing this approach, the agent-based AAS can identify and prevent problems and make decisions to stop its operations in order to avoid downtimes or potential issues that could compromise asset integrity. In such situation, the resource notifies all products with planned operations about canceling those operations, ensuring that all participants are aware of the situation and can adjust their plans accordingly.

Figure 18 (left) presents the behavior of the agent-based AAS representing a product. This behavior focuses on waiting for notifications to make decisions based on the received information. In this case, the agent-based AAS subscribes to specific topics where the information will be published. The topic is defined in the design phase and included in a submodel. As the notified information may cover a wide range of subjects, it is important to develop proper treatment and actions for each case. In this scenario, the subject involves problems related to failures. Therefore, specific actions are

performed in those conditions. For instance, when a product is notified about a resource failure, the product needs to find another resource capable of performing the same operation. This process can be done by searching for services provided by the resources (similar to the negotiation scenario). If one or more resources are available to provide this service, the product can negotiate with the resources or request the operation from a specific one. After the product reschedule its production plan, the process can continue as if it were an operation execution scenario.

As depicted in Figure 7, the exchange of information between the agent-based AASs of the unexpected event scenario is based on the FIPA Subscribe Interaction protocol [54] but with the inclusion of a broker entity. This interaction is based on the publish-subscribe schema, where the exchange of messages is topic-based and managed by a broker, which facilitates the communication between an entity that needs to communicate the same information to various others. In this interaction protocol, the products assume the role of ''Subscriber'' and the resources assume the role of ''Publisher''. The ''Subscriber'' subscribes to specific topics to receive notification from the ''Publisher'' when some unexpected event occurs.

Note that the protocol depicted in Figure 7 specifically outlines the interaction for event notification. In this regard, the protocols shown in Figures 5 and 6 can be used to facilitate the subsequent processes following the event notification. For example, these protocols support activities such as searching and negotiating with resources, as well as requesting the operation of a resource.

VIII. DISCUSSIONS

Based on the developed solution, this section discusses how the implemented agent-based AAS approach fulfills the key characteristics of an AAS Type 3, summarizes the main insights and experiences derived from the implementation, and outlines the identified limitations and future perspectives.

A. ASSET ADMINISTRATION SHELL TYPE 3 MINIMUM REQUIREMENTS

This subsection discusses how the implemented agent-based AAS approach meets the requirements to be considered an AAS Type 3 solution, which includes autonomy, intelligence, and collaborative capabilities, as previously described (see definitions in Section II). It is important to note that the level of each requirement can vary depending on the application goals and is highly influenced by the adopted algorithm/technique, i.e., incorporating advanced AI techniques, such as ML, Deep Learning (DL), and other sophisticated algorithms, can provide a higher degree of autonomy, intelligence, and collaboration. However, this approach does not focus on the specifics of individual algorithms but on designing components that meet these requirements. Moreover, since the proposed approach is modular, replacing algorithms is relatively straightforward. In this sense, different AI techniques or strategies can be implemented as needed to optimize autonomy, improve intelligence for decision-making, and reinforce collaboration in dynamic environments.

1) AUTONOMY

In this work, autonomy does not mean absolute freedom to decide arbitrarily but rather the capability to make decisions based on predefined criteria. For instance, in the operation execution scenario, despite the products following a predefined production plan, they independently make decisions and interact with resources to complete the plan without external intervention, ensuring that the system functions effectively while respecting predefined constraints. On the other hand, resources receive the product's requests and autonomously adjust their parameters to meet the product specifics.

Other examples include negotiation and unexpected event scenarios. In the negotiation scenario, assets in the system negotiate with each other based on predefined criteria to achieve both beneficial results. This negotiation process is autonomous since each asset makes decisions independently, following defined rules and protocols without external control.

In the unexpected event scenario, the system's autonomy is demonstrated by its ability to reschedule production plans in response to disturbances, such as equipment failures. In this sense, the assets through AAS Type 3 can autonomously assess the situation and reallocate resources to mitigate the impact. This ability to respond to condition changes without human intervention demonstrates the system's capability to operate autonomously within the boundaries of its predefined criteria.

2) INTELLIGENCE

In the proposed approach, intelligence can have different levels, e.g., ranging from basic rule-based mechanisms to advanced AI techniques. Basic rule-based mechanisms involve simple decision-making processes based on predefined rules and logic. On the other hand, advanced AI techniques include ML, DL, and other sophisticated algorithms that enable the system to learn from data, adapt to new situations, and improve over time. For instance, in the unexpected event scenario, the monitoring behavior of the agent-based AAS is based on a rule-based algorithm that analyses operational data from the asset to identify possible problems that can lead to downtimes or potential issues that could compromise the integrity of the asset.

Although this work does not address advanced AI techniques (e.g., ML algorithms and data analysis) to achieve intelligence, the intelligent behavior of the system emerges mainly from the interaction and collaboration among a group of distributed and autonomous agent-based AASs endowed with simple behaviors and/or limited intelligence. This collective intelligence provides an alternative approach to designing intelligent systems, where the traditional centralized control is replaced by a distributed functioning in which

the interactions between individuals result in the emergence of intelligent global behavior.

3) COLLABORATION

Unlike simple communication, collaboration goes beyond the exchange of information and comprises a deeper, interdependent process where all participants play crucial roles in achieving a common goal. In this regard, the implementation of the proposed approach in the different scenarios demonstrated different types of collaboration between the agent-based AASs.

In the operation execution scenario, assets collaborate by interacting in real time to perform operations and adjust production parameters dynamically. This type of collaboration is critical for maintaining efficiency and responsiveness in dynamic production environments, which involves exchanging data on current production status and requirements and coordinating actions to ensure smooth operation without human intervention.

In the autonomous negotiation scenario, assets negotiate with each other to autonomously reach agreements or decisions. This form of collaboration is essential for optimizing resource utilization and involves proposing and evaluating different proposals and reaching a consensus or making decisions without direct human oversight.

In the unexpected event scenario, assets play a crucial role in responding to unforeseen circumstances or disruptions. This type of collaboration is essential for maintaining resilience and minimizing the impact of unexpected events on production, which involves detecting and recognizing unexpected events and coordinating responses and actions among multiple assets.

B. CRITICAL REFLECTIONS ON DESIGN AND IMPLEMENTATION

Some of the experiences and reflections on the agent-based AAS approach's design and experimental implementation are presented in the following.

As an AAS-based solution, the proposed approach reflects the asset in the information world in the five upper layers of the RAMI4.0 model. This approach enhances traditional AAS capabilities through integration with agents. The agent implements the intelligent part of the agent-based AAS approach and is supported by standards, namely IEEE 2660.1, VDI/VDE 2193, and FIPA. The information part of the proposed approach comprises several submodels that structure the asset information in a standardized manner. Most submodels used in this work are official submodels defined by the IDTA, which ensures greater interoperability. However, this work also develop some submodels that comply with the AAS specifications.

The agent-based AAS approach decentralizes intelligence through autonomous, intelligent, and collaborative components that can employ multiple communication protocols to facilitate different types of interactions with each other, enabling a distributed problem-solving method. By leveraging the capabilities of autonomous agents, this approach enables more efficient handling of complex and dynamic environments. Each agent, with its specific knowledge and skills, contributes to the collective intelligence of the system, facilitating a cooperative effort towards achieving common goals.

The agent-based AAS has a modular structure, enabling different modules (behaviors, communication protocols, submodels) to be developed and integrated independently. This modularity allows easier maintenance, updates, and customization to suit various requirements. In this work, certain behaviors, communication protocols, and submodels were added, modified, or replaced depending on the specific scenario, demonstrating that the proposed approach can easily adapt to meet the required conditions.

The implemented agent-based AAS approach uses the submodels as information source. In this sense, each agent-based AAS can interpret the information from its respective asset's submodel to make decisions over time. Although the current implementation does not involve advanced learning algorithms, the ability of agents to interpret and act using information from the submodels demonstrates a certain level of intelligence in decision-making. Moreover, as previously discussed, since it is a modular approach, advanced learning algorithms can be easily integrated, enhancing the agent-based AAS's decision-making capabilities.

Through the process of design and implementation of the proposed approach in different scenarios, the reusability of the proposed approach was noticed, i.e., it became evident that several classes and submodels could be reused to meet various scenarios. Most of the developed classes were reused for the different scenarios, with customization only being necessary for classes that incorporate interaction protocols or are specific to particular behaviors. However, integrating these tailored classes is straightforward and only requires minor adjustments. Regarding the submodels, only their content needed to be modified, while the structure remained unchanged. Furthermore, the proposed approach allows system scalability since a designed agent-based AAS can be reused for several assets. For instance, the template, consisting of a common set of classes and submodels developed for an agent-based AAS designed for a product, can also be effectively applied to other similar products. This reusability extends beyond products to encompass various types of assets in the system.

After integrating an agent-based AAS into the system, the immediate recognition and seamless interaction with other entities are enabled through several key mechanisms, namely discovery mechanisms, standardized communication protocols, self-describing data in submodels, and decision-making capabilities. These elements work together to ensure that the new AAS can integrate, interact, and collaborate with existing entities without delay. The discovery mechanisms allow the system to automatically identify and connect with new agent-based AASs, while the use of standardized protocols ensures seamless communication. Self-describing submodels provide the necessary contextual information

for effective collaboration, and the AAS's decision-making capabilities enable autonomous adjustments and real-time coordination with other entities.

The unexpected event scenario provided an opportunity to analyze the robustness aspects of the implemented solution. This was evident when, at a specific time, one of the assigned resources for carrying out one of the operations experienced a failure, requiring the products to reschedule their production plans in order to complete the production process. In this context, the production process continues to function despite component failures. Although this condition is possible thanks to the redundancy of resources, the fact that the system can autonomously reorganize to continue operating demonstrates the system's robustness.

The agent-based AAS can be designed for assets placed in different hierarchy levels, such as the shop floor and business levels. While this work presented implementation examples at the shop floor level, particularly focusing on the horizontal collaboration between products and resources, the proposed approach can also enable vertical collaboration. This vertical collaboration is enabled since the agent-based AASs use standardized information models to structure information (submodels) and standardized communication protocols, data formats, and structures to facilitate seamless data exchange between components. In this scenario, an agent-based AAS representing a MES or ERP system could enable exchanging information and collaboration with shop floor devices.

The proposed approach can be implemented using several open-source development platforms to develop the intelligent (agent) and information (AAS submodels) parts of the agent-based AAS. For instance, in this work, the adopted platforms include JADE, Eclipse AASX Package Explorer, and AASX Server due to their efficiency and reliability in building agents and AAS submodels. Most of these platforms, including those used in this work, are compatible with various operating systems and can be deployed on different devices, ranging from single-board computers like Raspberry Pi to more powerful industrial computers. As a result, the designed agent-based AAS solution can be readily deployed in diverse scenarios involving hardware and software infrastructure variations.

C. LIMITATIONS AND FUTURE PERSPECTIVES

Although the proposed agent-based AAS approach demonstrated the feasibility of implementing AAS Type 3, several limitations and improvement opportunities were identified during its design and experimentation. This subsection aims to provide an overview of these aspects, highlighting architectural, operational, and semantic constraints that emerged during implementation, while also outlining potential directions for advancing the development and practical realization of AAS Type 3 systems.

This work does not cover semantics at any level despite their importance in achieving interoperability. For example, ECLASS and IEC Common Data Dictionary are two dictionaries of standardized semantics based on the IEC 61360 standard, which could be used to describe asset properties in an unambiguous, machine-readable, and industry-independent manner.

The error analysis was performed only qualitatively by monitoring message exchanges and execution logs to verify behavioral correctness and synchronization stability among distributed agents. Minor timing deviations, typically within a few milliseconds, were observed due to network latency but did not affect coordination accuracy. The results confirm the stability and reliability of the proposed implementation, which successfully demonstrated consistent behavior across all experimental runs.

The current implementation assumes a trusted execution environment with local network communication among agents, without specific mechanisms for data integrity, authentication, or encryption, which would be essential in global shared production systems. In this sense, future extensions could incorporate data space architectures [63], such as those defined by the International Data Spaces (IDS) initiative, to enable secure and policy-compliant data exchange across organizational boundaries. Furthermore, the experimental validation was conducted in a small-scale emulated setup, and scalability to industrial-level applications may require further optimization of communication overhead and distributed synchronization. Nevertheless, these aspects do not limit the proof of concept or the validity of the proposed approach, as the current implementation successfully demonstrates the feasibility and core mechanisms required for realizing AAS Type 3 through the proposed agent-based approach.

A relevant discussion concerns the degree of integration between the agent and the AAS submodel management. In the current design, a loosely coupled architecture was adopted to preserve modularity and compatibility with existing AAS Type 1/2 tools. However, as highlighted in recent discussions, a fully ''agentified'' AAS Type 3 could evolve toward a model where the agent itself directly manages its AAS submodels, effectively merging the intelligence and administrative layers. This perspective opens the possibility for the agent to possess its own capabilities and skills, which could be represented alongside asset-level properties within the AAS information model. Such approach would move toward an indivisible integration between the asset and its intelligent intermediary, representing a promising direction for the next generation of AAS Type 3 implementations.

Another important aspect concerns the operation and management of distributed AAS Type 3 systems. Beyond dashboard-based supervision, future implementations will require a DevOps-oriented approach encompassing the encapsulation of AAS Type 3 into containers, their deployment across nodes with specific hardware and communication constraints (e.g., CPU/RAM, I/O buses, network latency), and their lifecycle management including scalability, redundancy, and fault tolerance. These requirements point to the need for orchestration frameworks specifically adapted to the AAS Type 3 solution.

Finally, the ongoing advance of LLMs open promising opportunities for extending the cognitive capabilities of agent-based AASs. Their integration can enable contextual reasoning, adaptive decision-making, and more natural interaction with human operators. By leveraging the language understanding and generalization capabilities of LLMs, agent-based AASs could interpret unstructured industrial data, generate semantically enriched insights, and support explainable and human-aligned decision processes.

IX. CONCLUSION

This article presented an agent-based approach for implementing the concept of AAS Type 3, addressing the current lack of specifications and guidelines for realizing autonomous, intelligent, and collaborative I4.0 components. The proposed solution integrates MAS into the AAS framework, combining the standardized information structure of the AAS with the decision-making and collaborative capabilities of software agents.

Through designing and experimental implementation, the approach demonstrated how AAS Type 3 can be effectively realized by extending traditional AAS functionalities beyond data representation and toward proactive and adaptive behaviors, enabling seamless interaction between assets and supporting both horizontal and vertical collaboration within the I4.0 ecosystem.

The experimental results confirmed the feasibility and flexibility of the proposed approach in supporting distributed coordination, autonomous negotiation, and dynamic reconfiguration in manufacturing scenarios. Additionally, the article discussed the experiences and insights gained during the design and experimental implementation, as well as limitations and future perspectives.

In summary, this work contributes to advancing the practical realization of AAS Type 3 by demonstrating how intelligence, autonomy, and collaboration can be embedded within the AAS through an agent-based approach, paving the way for more adaptive, self-organizing, and resilient industrial environments.

REFERENCES

[1] A. W. Colombo, S. Karnouskos, O. Kaynak, Y. Shi, and S. Yin, "Industrial cyberphysical systems: A backbone of the fourth industrial revolution," *IEEE Ind. Electron. Mag.*, vol. 11, no. 1, pp. 6–16, Mar. 2017.

[2] *Reference Architecture Model Industrie 4.0 (RAMI4.0) English Translation*, document SPEC 91345, DIN, 2016.

[3] *Details of the Asset Administration Shell—From Idea to Implementation*, Plattform Industrie 4.0, Berlin, Germany, 2018.

[4] L. Sakurada, F. de la Prieta, and P. Leitao, "Ten years of asset administration shell: Developments, research opportunities, and adoption challenges," *IEEE Access*, vol. 13, pp. 127721–127741, 2025.

[5] *Structure of the Administration Shell—Continuation of the Development of the Reference Model for the Industrie 4.0 Component*, Plattform Industrie 4.0, Berlin, Germany, 2016.

[6] *Implementation Strategy Industrie 4.0—Report on the Results of the Industrie 4.0 Platform*, Bitkom, VDMA and ZVEI, 2016.

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

[8] L. Sakurada, F. De la Prieta, and P. Leitao, "The role of multi-agent systems in realizing asset administration shell type 3," *Future Internet*, vol. 17, no. 7, p. 270, Jun. 2025.

[9] C. Wagner, J. Grothoff, U. Epple, R. Drath, S. Malakuti, S. Grüner, M. Hoffmeister, and P. Zimermann, "The role of the Industry 4.0 asset administration shell and the digital twin during the life cycle of a plant," in *Proc. 22nd IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2017, pp. 1–8.

[10] T. A. Abdel-Aty, E. Negri, and S. Galparoli, "Asset administration shell in manufacturing: Applications and relationship with digital twin," *IFAC-PapersOnLine*, vol. 55, no. 10, pp. 2533–2538, 2022.

[11] J. Zhang, C. Ellwein, M. Heithoff, J. Michael, and A. Wortmann, "Digital twin and the asset administration shell," *Softw. Syst. Model.*, vol. 24, no. 3, pp. 1–23, Jun. 2025.

[12] *Automation Systems and Integration—Digital Twin Framework for Manufacturing—Part 1: Overview and General Principles*, Standard ISO 23247, 2021.

[13] *Automation Systems and Integration—Digital Twin Framework for Manufacturing—Part 2: Reference Architecture*, Standard ISO 23247, 2021.

[14] *Digital Twin Capabilities Periodic Table*, Digital Twin Consortium, Boston, MA, USA, 2022.

[15] A. Köcher, A. Belyaev, J. Hermann, J. Bock, K. Meixner, M. Volkmann, M. Winter, P. Zimmermann, S. Grimm, and C. Diedrich, "A reference model for common understanding of capabilities and skills in manufacturing," *At-Automatisierungstechnik*, vol. 71, no. 2, pp. 94–104, Feb. 2023.

[16] *Verwaltungsschale in Der Praxis*, Plattform Industrie 4.0, Berlin, Germany, 2020.

[17] *Specification of the Asset Administration Shell—Part 1: Metamodel (version 3.0.1)*, Industrial Digital Twin Association, Frankfurt am Main, Germany, 2024.

[18] *Specification of the Asset Administration Shell—Part 2: Application Programming Interfaces (version 3.0.3)*, Industrial Digital Twin Association, Frankfurt am Main, Germany, 2024.

[19] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *Knowl. Eng. Rev.*, vol. 10, no. 2, pp. 115–152, Jun. 1995.

[20] S. Cavalieri and M. G. Salafia, "A model for predictive maintenance based on asset administration shell," *Sensors*, vol. 20, no. 21, p. 6028, Oct. 2020.

[21] Y. Xia, M. Shenoy, N. Jazdi, and M. Weyrich, "Towards autonomous system: Flexible modular production system enhanced with large language model agents," in *Proc. IEEE 28th Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2023, pp. 1–8.

[22] S.-J. Shin and J. Um, "Deploying data analytics models in asset administration shells: Energy prediction in manufacturing," *Eng. Appl. Artif. Intell.*, vol. 138, Dec. 2024, Art. no. 109269.

[23] J. R. Rahal, A. Schwarz, B. Sahelices, R. Weis, and S. D. Antón, "The asset administration shell as enabler for predictive maintenance: A review," *J. Intell. Manuf.*, vol. 36, no. 1, pp. 19–33, Jan. 2025.

[24] A. Sidorenko, M. Volkmann, W. Motsch, A. Wagner, and M. Ruskowski, "An OPC UA model of the skill execution interaction protocol for the active asset administration shell," *Proc. Manuf.*, vol. 55, pp. 191–199, May 2021.

[25] J. Arm, T. Benesl, P. Marcon, Z. Bradac, T. Schröder, A. Belyaev, T. Werner, V. Braun, P. Kamensky, F. Zezulka, C. Diedrich, and P. Dohnal, "Automated design and integration of asset administration shells in components of Industry 4.0," *Sensors*, vol. 21, no. 6, p. 2004, Mar. 2021.

[26] M. Simon, S. Jungbluth, A. Farrukh, M. Volkmann, J. Hermann, and M. Ruskowski, "Implementation of asset administration shells in a shared production scenario with Gaia-X," in *Proc. IEEE 28th Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2023, pp. 1–8.

[27] S. Grunau, M. Redeker, D. Göllner, and L. Wisniewski, "The implementation of proactive asset administration shells: Evaluation of possibilities and realization in an order driven production," in *Technologien Für Die Intelligente Automation*, 2022, pp. 131–144.

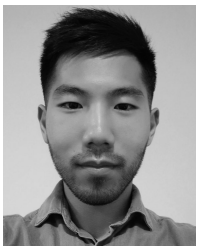
[28] K. Kanaan, J. Wermann, M. A. Bär, and A. W. Colombo, "Industry 4.0-compliant digitalization of a re-configurable and flexible laser cutter module within a digital factory," in *Proc. IEEE Int. Conf. Ind. Technol. (ICIT)*, Apr. 2023, pp. 1–7.

[29] M. Stolze, A. Belyaev, C. Kosel, C. Diedrich, and A. Barnard, "Realizing automated production planning via proactive AAS and business process models," in *Proc. IEEE 29th Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2024, pp. 1–8.

[30] X. Ye and S. H. Hong, "Toward Industry 4.0 components: Insights into and implementation of asset administration shells," *IEEE Ind. Electron. Mag.*, vol. 13, no. 1, pp. 13–25, Mar. 2019.

[31] X. Ye, J. Jiang, C. Lee, N. Kim, M. Yu, and S. H. Hong, "Toward the plug-and-produce capability for Industry 4.0: An asset administration shell approach," *IEEE Ind. Electron. Mag.*, vol. 14, no. 4, pp. 146–157, Dec. 2020.

[32] X. Ye, S. H. Hong, W. S. Song, Y. C. Kim, and X. Zhang, "An Industry 4.0 asset administration shell-enabled digital solution for robot-based manufacturing systems," *IEEE Access*, vol. 9, pp. 154448–154459, 2021.
[33] Q. Zhou, Y. Wu, C. Gu, W. Meng, S. He, and Z. Shi, "AASPMP: Design and implementation of production management platform based on AAS," in *Proc. IEEE 20th Int. Conf. Ind. Informat. (INDIN)*, Jul. 2022, pp. 515–520.
[34] B. Vogel-Heuser, F. Ocker, and T. Scheuer, "An approach for leveraging digital twins in agent-based production systems," *At-Automatisierungstechnik*, vol. 69, no. 12, pp. 1026–1039, Dec. 2021.
[35] S. Jungbluth, J. Hermann, W. Motsch, M. Pourjafarian, A. Sidorenko, M. Volkmann, K. Zoltner, C. Plociennik, and M. Ruskowski, "Dynamic replanning using multi-agent systems and asset administration shells," in *Proc. IEEE 27th Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Stuttgart, Germany, Sep. 2022, pp. 1–8.
[36] A. López, O. Casquero, and M. Marcos, "Design patterns for the implementation of industrial agent-based AASs," in *Proc. 4th IEEE Int. Conf. Ind. Cyber-Phys. Syst. (ICPS)*, May 2021, pp. 213–218.
[37] L. Sakurada, F. D. L. Prieta, and P. Leitao, "A methodology for integrating asset administration shells and multi-agent systems," in *Proc. IEEE 32nd Int. Symp. Ind. Electron. (ISIE)*, Jun. 2023, pp. 1–6.
[38] A. Sidorenko, W. Motsch, M. van Bekkum, N. Nikolakis, K. Alexopoulos, and A. Wagner, "The MAS4AI framework for human-centered agile and smart manufacturing," *Frontiers Artif. Intell.*, vol. 6, pp. 1–20, Sep. 2023.
[39] A. T. Bernhard, S. Jungbluth, A. Karnoub, A. Sidorenko, W. Motsch, A. Wagner, and M. Ruskowski, *I4.0 Holonic Multi-Agent Testbed Enabling Shared Production*. Cham, Switzerland: Springer, 2024, pp. 231–250.
[40] V. Siatras, E. Bakopoulos, P. Mavrothalassitis, N. Nikolakis, and K. Alexopoulos, "Production scheduling based on a multi-agent system and digital twin: A bicycle industry case," *Information*, vol. 15, no. 6, p. 337, Jun. 2024.
[41] P. Rübel, W. Motsch, A. Bernhard, S. Jungbluth, and M. Ruskowski, "Agent-based communication for fault diagnosis in skill-based production environments using messages based on i4.0 language and asset administration shells," in *Advances in Artificial Intelligence in Manufacturing II*. Cham, Switzerland: Springer, 2025, pp. 157–170.
[42] *VDI/VDE 2193 Part 1—Language for I4.0 Components—Structure of Messages*, VDI/VDE, Düsseldorf, Germany, 2020.
[43] *VDI/VDE 2193 Part 2—Language for I4.0 Components—Interaction Protocol for Bidding Procedures*, VDI/VDE, Düsseldorf, Germany, 2020.
[44] Ind. Digit. Twin Assoc. *Repository Submodel Templates—Capability (Version 1.0)*. Accessed: Apr. 5, 2024. [Online]. Available: https://github.com/admin-shell-io/submodel-templates/tree/main/development/Capability/1/0
[45] *Modelling the Semantics of Data of an Asset Administration Shell With Elements of ECLASS*, Plattform Industrie 4.0 and ECLASS, Berlin, Germany, 2021.
[46] *IEC 61360-4—IEC/SC 3D—Common Data Dictionary (CDD-V2.0018.0001)*. Accessed: Apr. 16, 2024. [Online]. Available: https://cdd.iec.ch/cdd/iec61360/iec61360.nsf
[47] *IDTA 02015-1-0—Control Component Type*, Ind. Digital Twin Assoc., Frankfurt am Main, Germany, 2023.
[48] *IDTA 02016-1-0—Control Component Instance*, Ind. Digital Twin Assoc., Frankfurt am Main, Germany, 2023.
[49] *IDTA 02017-1-0 Asset Interfaces Description*, Ind. Digital Twin Assoc., Frankfurt am Main, Germany, 2024.
[50] *IDTA 02058-1-0 Artificial Intelligence Dataset*, Ind. Digital Twin Assoc., Frankfurt am Main, Germany, 2025.
[51] *IDTA 02060-1-0 Artificial Intelligence Model Nameplate*, Ind. Digital Twin Association, Frankfurt am Main, Germany, 2025.
[52] *IDTA 02059-1-0 Artificial Intelligence Deployment*, Industrial Digital Twin Assoc., Frankfurt am Main, Germany, 2025.
[53] FIPA. (2002). *FIPA Specifications*. [Online]. Available: http://www.fipa.org/specifications/index.html
[54] *FIPA Subscribe Interaction Protocol Specification*, FIPA, Geneva, Switzerland, 2002.
[55] A. Lopez, L. Sakurada, P. Leitao, O. Casquero, E. Estevez, F. De la Prieta, and M. Marcos, "Technology-independent demonstrator for testing Industry 4.0 solutions," in *Proc. IEEE 20th Int. Conf. Ind. Informat. (INDIN)*, Jul. 2022, pp. 21–26.
[56] P. Leitão, "An agile and adaptive holonic architecture for manufacturing control," Ph.D. thesis, Fac. Eng. Univ. Porto, 2004.
[57] *Aspects of the Research Roadmap in Application Scenarios*, Plattform Industrie 4.0, ECLASS, Berlin, Germany, 2016.
[58] Z. Wrona, W. Buchwald, M. Ganzha, M. Paprzycki, F. Leon, N. Noor, and C.-V. Pal, "Overview of software agent platforms available in 2023," *Information*, vol. 14, no. 6, p. 348, Jun. 2023.
[59] IDTA. (2023). *DotAAS Part 2 | HTTP/REST | AAS Repository Service Specification*. [Online]. Available: https://app.swaggerhub.com/apis/Plattformi40/AssetAdministrationShellRepositoryServiceSpecification/V3.0.1SSP-001
[60] *IEEE Recommended Practice for Industrial Agents: Integration of Software Agents and Low-Level Automation Functions*, Standard IEE 2660.1-2020, 2021.
[61] *FIPA Request Interaction Protocol Specification*, FIPA, Geneva, Switzerland, 2002.
[62] L. S. Nelson, "The shewhart control chart—Tests for special causes," *J. Quality Technol.*, vol. 16, no. 4, pp. 237–239, Oct. 1984.
[63] *Data Connector Report*, Int. Data Spaces Assoc., Dortmund, Germany, 2024.



LUCAS SAKURADA received the degree in electronic engineering from the Federal University of Technology—Paraná (UTFPR), Brazil, and the M.Sc. degree in industrial engineering from the Polytechnic Institute of Bragança (IPB), Portugal, in 2019. He is currently pursuing the Ph.D. degree in computer engineering with the University of Salamanca (USAL), Spain. Since 2019, he has been a Researcher with the Research Centre in Digitalization and Intelligent Robotics (CeDRI), Portugal. His research interests include the field of industrial digitization and engineering of industry 4.0-compliant solutions, particularly using asset administration shells, multi-agent systems, and industrial cyber-physical systems.

FERNANDO DE LA PRIETA is currently a Full Professor with the Department of Computer Science and Automation, University of Salamanca, where he is also a Rector's Delegate for Digital Transformation. He is also a member of the Bioinformatics, Intelligent Systems, and Educational Technology (BISITE) Research Group. As a Researcher, he has followed a clear line of research, focused on the integration of organizational multi-agent systems, machine learning, and advanced architectures in numerous fields. He has more than 50 publications in international journals (many of them indexed according to the JCR index). Additionally, he has published more than 100 articles in books and international conferences (some of them indexed in the CORE ranking). He has worked in more than 100 research projects (12 as the Principal Investigator), 16 of them were international. As a result of his work, 38 intellectual properties have been registered. He has also taken an active part in the organization of international conferences, some of them included in the CORE ranking: IEEE-GLOBECOM (core B), ICCBR (Core B), CEDI, PAAMS (core C), ACM-SAC (core B), IEEE-FUSION (core C), and others.

PAULO LEITAO (Senior Member, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of Porto, Porto, Portugal, in 2004.

He is currently a Full Professor with the Department of Electrical Engineering, Polytechnic Institute of Bragança, Bragança, Portugal, and the Coordinator of the Research Centre in Digitalization and Intelligent Robotics (CeDRI). He has authored or co-authored ten books and more than 350 papers in high-ranked international scientific journals and conference proceedings (peer-reviewed) and has coordinated/participated in several national and international research projects and networks of excellence. His research interests include intelligent and reconfigurable systems, industrial cyber-physical systems, multi-agent systems, digital twins, and factory automation. He is a Senior Member of the IEEE Industrial Electronics Society (IES) and the Systems, Man and Cybernetics Society (SMCS), the Past Chair of the IEEE IES Technical Committee on Industrial Agents, and the Chair of the established IEEE 2660.1-2020 Standard.