Methods

Birgit Vogel-Heuser, Felix Ocker* and Tobias Scheuer

An approach for leveraging Digital Twins in agent-based production systems

Ein Ansatz zur effizienten Erstellung agentenbasierter Produktionssysteme mittels Digitaler Zwillinge

https://doi.org/10.1515/auto-2021-0081
Received June 4, 2021; accepted September 7, 2021

Abstract: To cope with individualization and the high costs of downtimes, modern production systems should be flexible, adaptable, and resilient. Multi-Agent Systems are suitable to address these requirements by decentralizing production systems. However, the agent paradigm is still not widely applied. One of the key reasons is that the agents' knowledge bases had to be created manually, which is cumbersome, error-prone, and insufficiently standardized. Digital Twins have the potential to solve this issue, as they describe relevant information in a standardized way. This paper presents an approach to leveraging Digital Twins, i. e., the Asset Administration Shell, to realize Multi-Agent Systems in the production context. For this, a parser automatically extracts relevant information from the Digital Twins and initializes the individual agents in a Multi-Agent System, i. e., PADE.

Keywords: Digital Twins, Multi-Agent Systems, smart factories, Knowledge Base

Zusammenfassung: Um der Individualisierung und den hohen Kosten von Stillstandszeiten gerecht zu werden, müssen moderne Produktionssysteme flexibel, anpassungsfähig und resilient sein. Multi-Agenten Systeme sind geeignet, um diesen Herausforderungen zu begegnen, indem sie moderne Produktionssysteme dezentralisieren. Allerdings ist das Agenten-Paradigma noch nicht weit verbreitet. Einer der Hauptgründe dafür ist, dass die Wissensbasen der Agenten bisher manuell erstellt werden, was umständlich, fehleranfällig und unzureichend standardisiert ist. Digitale Zwillinge haben das Potenzial, dieses Problem zu lösen, da sie standardisierte Beschreibungen relevanter Informationen bereitstellen. In diesem Beitrag wird ein Ansatz zur Nutzung Digitaler Zwillinge, speziell der Verwaltungsschale, vorgestellt, um Multi-Agent Systeme im Produktionskontext zu realisieren. Dafür extrahiert ein Parser automatisiert relevante Informationen aus Digitalen Zwillingen und initialisiert die einzelnen Agenten mit dem Multi-Agenten Framework PADE.

Schlagwörter: Digitale Zwillinge, Multi-Agenten Systeme, Intelligente Fabriken, Wissensbasis

***Corresponding author: Felix Ocker,** Institute of Automation and Information Systems, Department of Mechanical Engineering, TUM School of Engineering and Design, Technical University of Munich, Munich, Germany, e-mail: felix.ocker@tum.de
Birgit Vogel-Heuser, Institute of Automation and Information Systems, Department of Mechanical Engineering, TUM School of Engineering and Design, Core Member of MDSI and Member of MIRMI, Technical University of Munich, Munich, Germany, e-mail: vogel-heuser@tum.de
Tobias Scheuer, Institute of Automation and Information Systems, Department of Mechanical Engineering, TUM School of Engineering and Design, Technical University of Munich, Munich, Germany, e-mail: tobias.scheuer@tum.de

1 Motivation

Modern manufacturing systems face significant challenges resulting from globalized and customer-driven markets as well as an ongoing acceleration of technological development. Shorter product life cycles and customization are essential trends that make increased flexibility and manufacturing concepts necessary [1]. To cope with these challenges, production systems have to evolve into smart factories. One promising technology are Multi-Agent Systems (MASs). Here, all the resources, i. e., machines, and products within the production system are represented by autonomous intelligent entities. These intelligent agents communicate and negotiate to reach their individual goals, which are aligned with the companies' goal to create products for their customers. To do so, the intelligent agents require information about their environment, their abilities, their needs, and their goals. This information is stored in each agent's Knowledge Base (KB). Even though several sound agent frameworks are available, the creation of the agents' KBs remains a major

challenge that impedes the adoption of MASs. This is because the creation of such KBs is cumbersome, and they should be created in a standardized way to support communication between the agents and thus interoperability. Digital Twins (DTs) are a possibility to provide the needed information and standardized structure [2].

This paper presents an approach for leveraging DTs for MASs in the context of production systems. Here, we focus three research questions. First, it has to be analyzed which information is required by the different agents and what can and cannot be provided by DTs *(RQ1)*. Second, it needs to be researched how a MAS can be created from these DTs *(RQ2)*. Third, the operation of the DT-based MAS needs to be investigated *(RQ3)*.

The rest of the paper is structured as follows. Section 2 gives an overview of related literature, while Section 3 presents the framework for creating a MAS based on DTs. Section 4 describes a demonstrator and the prototypical implementation, which is evaluated using a simulation. Section 4 also discusses the framework's opportunities and limitations, and the paper concludes with a summary and an outlook in Section 5.

2 Related literature

This section provides an overview of related work regarding established MAS, DTs, and how the two concepts can be combined.

2.1 Multi-Agent Systems

MASs aim to support decentralized control and come in many different forms, thus adapting to the individual application. Vogel-Heuser et al. [3] define MASs as a "computational paradigm introduced in the distributed artificial intelligence field, characterized by the decentralization and parallel execution of activities based on autonomous agents". The underlying agents are intelligent entities, which are characterized by their autonomy, their encapsulation, and their goals [4]. For this, agents usually rely on believes regarding their environment, desires, i. e., their goals, and intentions, i. e., the actions they have committed themselves to, resulting in the term Believes Desires Intentions (BDI) agent. Similarly, the VDI [5] describes an agent as "an encapsulated hardware or software entity with specified objectives regarding the control of a technical system (or part thereof)". MASs replace centralized control architectures with distributed decision-making by the agents, which enables the production system to adapt itself to changing environments and tasks autonomously [3].

In order to realize MASs in the production context, several types of agents are oftentimes used. Cruz et al. [6] classified these agent types and summarized them as patterns. They conclude that most MASs include the agent types *Agent Management System (AMS)*, *Communication Agent*, *Resource Agent*, and *Process Agent* [6, 7]. An AMS keeps track of all active and connected agents, as well as their communication addresses, while Communication Agents (CAs) act as interfaces for the MAS to legacy systems. Resource Agents (RAs) represent manufacturing resources, including transportation resources. A RA "drives its resource (actuation) and keeps its internal state model synchronized with the resource through appropriate input (sensor readings)" [8]. Process Agents are responsible for coordinating the production processes by monitoring them, e. g., using co-simulations. A related type of agent is the Product Agent (PA), which is a "software component that is responsible for fulfilling production requirements for an associated physical part through interactions with other agents in the system" [9]. Its objective is to request and schedule operations from the system's resources based on the customer specification [9]. Two types of PAs can be distinguished [9]. First, there are rule-based PAs, which are used, e. g., in PROSA [8], PABADIS [10], ADMARMS [11], or ADACOR [12]. Second, there are model-based PAs, which are mostly implemented in the form of Finite State Machines [13]. Rule-based agents ensure timeliness of the decision-making process, which is crucial whenever there are real time requirements, e. g., on the field-level. However, rule-based intelligence is difficult to scale to larger, more complex systems as the ruleset's size is related to the number of interactions among agents. Model-based architectures on the other hand seem more adequate to address scalability, but there are less approaches available [9]. Various software frameworks have been developed for creating MASs. Prime examples are the established Java Agent Development Framework (JADE) [14] and the actor framework akka.[1] For both exist Python implementations, namely Python Agent Development Framework (PADE) [15] and Pykka,[2] respectively. Additionally, custom frameworks can be developed. In order to realize intelligent agents, several aspects have to be considered [16]. An agent's behavior is influenced by the *objec-*

1 https://akka.io/ last accessed: November 24, 2021.

2 https://pykka.readthedocs.io/en/stable/ last accessed: November 24, 2021.

tives it is trying to accomplish, which are also referred to as *desires*. Examples are providing and executing an ability, or reaching a certain destination. In case an agent has several objectives, they have to be integrated into a joint objective function. These objectives can be predefined, defined during run-time, or they may even change as the agent learns. In order to make informed decisions, information about the environment, oftentimes referred to as *believes*, is needed and saved in a so-called *environment model*. This includes information such as locations of the agent itself, but also its surroundings, potential transportation pathways, available sensor inputs, capabilities, and communication addresses of other resources. Lastly, all internal actions and interactions with its surroundings are described in an agent's *abilities*, e. g., all abilities to control subordinated machine parts, or the ability to manipulate another physical entity. Even though these KBs are crucial for MASs, their creation is still cumbersome, and there are no standardized ways for creating them, which is a prerequisite for interoperability.

2.2 Digital Twins

The uprising technology of DTs has the potential to provide standardized KBs for agents in MASs. A DT is understood as a "dynamic virtual representation of a physical object or system across its lifecycle" [17]. Such a virtual counterpart to a physical object, i. e., an *asset*, aims to capture all the asset's relevant information, including its abilities and status. Thus, DTs present a promising possibility for real-time simulations and knowledge accumulation throughout the entire lifetime of a product [18]. Applications of DTs are diverse and can be categorized according the lifecycle phase, i. e., primarily design or operation. During design, DTs seem valuable for configuration, inconsistency management, and testing, but they may also be beneficial for decentralized operation and process optimization in later phases [19]. Further applications include production management and control, systems engineering, especially when simulation-based, scheduling optimization, and monitoring, e. g., for fault diagnosis [20]. Moyne et al. [21] also identified predictive maintenance, model-based process control, and real-time scheduling including dispatch as potential applications.

By now, there are various DT frameworks available. In the following, we briefly introduce three of them. The Asset Administration Shell (AAS) is being developed by the German Industry 4.0 Initiative [22]. It is a virtual representation of any I4.0 component in an I4.0 system. The AAS aims to ensure interoperability in an a priori way by relying on a rigorously standardized description of data, information, and services using a shared metamodel. That way, the AAS helps overcome the "information silo problem" [23], which means that information is kept in silos and cannot be used across systems. Key concepts included in the AAS metamodel include a class for assets, but also a top class for the AAS, i. e., the DT itself [22]. The AAS class aggregates different *submodels* and *submodel elements* to capture information about the asset in question. These descriptions can be refined, e. g., using the classes *property* and *capability*. Using the information captured, three different types of AASs can be realized [24]. The passive AAS is a file that provides a standardized way to exchange information associated with the asset. It is typically exchanged as an aasx file. Similarly, the reactive AAS is a passive AAS, but it is connected to the asset to ensure timeliness of data, and it can be accessed via an Application Programming Interface (API). Lastly, the proactive AAS also provides CRUD (create, read, update, delete) capabilities but can additionally communicate and bid with other AAS using the I4.0 language [24, 25]. The Digital Twin Definition Language (DTDL)[3] is a "language for describing models for IoT Plug and Play devices, device digital twins, and logical digital twins" developed by Microsoft. Compared to the AAS it is minimalist in its specification, but the DTDL still enables the semantic description of a DT's abilities. DTDL uses the JSON variant JSON-LD and relies on the six core classes *Interface*, *Telemetry*, *Property*, *Command*, *Relationship*, and *Component*. Additionally, there are custom approaches such as the DT architecture reference model Cloud-based Cyber-Physical Systems (C2PS) [26]. It focuses on the key CPS properties computation, control, and communication. C2PS uses a Bayesian network and fuzzy logic-based rule set for decision-making. Communication occurs either directly in the physical layer or through the cloud layer, but is not further specified [26].

2.3 Leveraging Digital Twins in Multi-Agent Systems

While MASs provide a suitable architecture to create decentralized and highly adaptable autonomous production systems, DTs have the potential to bring this vision into practice by providing standardized models, API access, and data exchange. Individually, there have been numerous advances in both fields. Also, some approaches that

3 https://github.com/Azure/opendigitaltwins-dtdl last accessed: November 24, 2021.

leverage both technologies have already been developed for other domains. In order to support spacecraft testing, Zhang et al. [27] developed a DT and MAS based architecture, which reduced the time spent on product test design and implementation by 20 %, and increased the precision of these by 15 %. For the farming sector, Laryukhin et al. [28] designed a Cyber-Physical System (CPS) using MASs and DTs to represent farms, plants, soil, and fertilizer to maximize yield. For logistic processes in CPSs in the postal industry, Niati at al. [29] developed a DT that consists of models created using the Business Process Modeling Notation and the Unified Modeling Language's Activity Diagrams. Using JADE, they derive a MAS that uses these behaviour models. However, they do not leverage current standardization efforts for DTs, such as the AAS. For the manufacturing domain, Zheng et al. [30] present an approach to quality assurance, which relies on a MAS and DTs. Even though they suggest using a top-level ontology for increasing interoperability, their approach relies on engineers using the ontology editor Protégé instead of building on a widely accepted format for DTs, such as the AAS or the DTDL. These heterogeneous approaches indicate the potential of the combination of DTs and MASs also for the production domain. This potential has also been theoretically confirmed in other related work [2, 31]. Vogel-Heuser et al. [2] gathered requirements for selected Industry 4.0 challenges, and analyzed industrial and research MAS applications. They conclude that MASs are a suitable technology to solve challenges imposed by Industry 4.0 and that AASs are a suitable supplement. However, no specific framework is provided that leverages DTs for MASs. Regarding the analysis of the AAS' suitability for capturing relevant information *(RQ1)*, we intend to extend our own theoretical prior work [31]. However, to the best of our knowledge, DTs and agents, specifically the AAS, have not yet been combined in the production domain. Hence, there is a need for a framework that supports engineers in leveraging DT for creating *(RQ2)* and operating *(RQ3)* MAS. Here, practical implementation challenges in the manufacturing context shall be identified and solved.

3 Approach for leveraging Digital Twins to create a Multi-Agent System

This section describes the assumptions made and gives an overview of the different agent types we consider in a production system, cp. Figure 1. For these agents, we delineate the tasks they execute, the information they require, and how this information is captured using a DT, specifically the AAS. Subsequently, we describe the recursive architecture developed. To adequately represent the hierarchical structure of a production system, each resource agent may also serve as a Directory Facilitator (DF), cp. Figure 1, for its subordinate resources. The section concludes with details regarding the agents' initialization process, and their decision-making process.

3.1 Assumptions

Within the context of this paper, we make several simplifying assumptions. Regarding the available DTs for resources, we assume that they are modularized sensibly as Distributed Digital Twins (DDTs) according to the production system's structure. Also, we assume that engineers specify these DDTs so that they include the relevant information for the respective physical entities, i. e., products and resources. For this, engineers may either rely on tools for DT creation such as the Asset Administration Shell exchange format (AASX) Package Explorer[4] or automate DT creation. Furthermore, we expect the product description to be available in the form of processes to be executed. This is a simplification, as products are usually described via features. However, this assumption may be lifted by relying on approaches for feature-process matching as used in prior work [32]. Regarding the layout, we assume simplified coordinates, i. e., resources and products are repre-

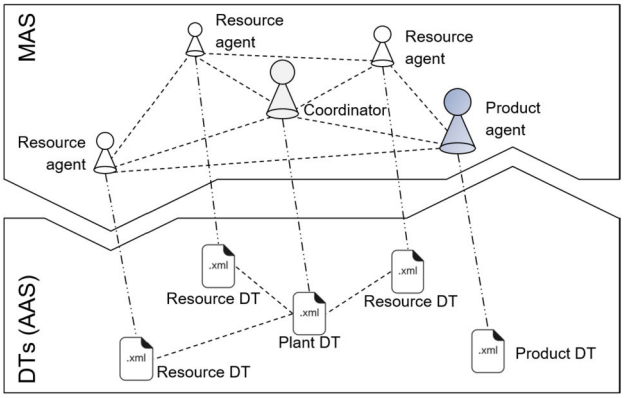

Figure 1: Framework for combining Digital Twins and Multi-Agent Systems.

4 https://github.com/admin-shell-io/aasx-package-explorer last accessed: November 24, 2021.

sented using 2D-point-coordinates. The resources are connected via fixed transport resources such as conveyors, so far we disregard automated guided vehicles, though. Even though these do not represent reality, the approach could be extended by leveraging more advanced layout checks [33]. Finally, this paper focuses on the combination of DTs and MASs. Hence, the resulting framework is evaluated using a simulation of the production system and the products. This allows us to ignore the connection of the individual agents to the respective physical entities. In practice, the agents may either be connected directly to the assets if only passive AAS are available, or they can be connected to their assets via their "reactive" DTs. In both cases, engineers may rely on state of the art technologies [34] for connecting the virtual entities to their physical counterparts.

3.2 Agent types, tasks, and relevant information

Within the context of this work, we focus on agents that represent resources and products. For both types of physical entities, there are usually DTs available, while, e. g., process DTs are less common. RAs are suitable to represent all kinds of resources, ranging from individual machines to entire plants. In all cases, a RA has to offer production capabilities to PAs upon request. Hereby, it has to take into account the physical resource's current status, e. g., if the capabilities are limited due to degradation. Additionally, the RA has to initiate the actual production processes by controlling the physical resource. In case a RA has subordinate RAs, it also serves as a DF. PAs on the other hand represent the products. Hence, they have to request the capabilities required for realizing the desired product features from RAs. To do this in a goal-oriented manner, they have to keep track of the product's status, and make decisions how to schedule the next steps within the production process. In order to fulfil these tasks, the agents require various pieces of information, ranging from a description of their capabilities for RAs to the production status for PAs. Table 1 gives an overview of the information required by RAs and PAs.

Fortunately, the AAS' metamodel explicitly provides a class for representing resource capabilities, to describe "implementation-independent description[s] of the potential of an asset to achieve a certain effect in the physical or virtual world" [22]. The more generic class *submodel element* can be used to describe and differentiate assets and is thus suitable to capture an agent's objective function or a process required by a product. Several submodel elements representing processes required can be aggregated using a *submodel collection*. For specifying single attributes, the AAS' metamodel provides properties and ranges. Together, these classes are sufficient to capture the information required by both RAs and PAs.

Table 1: AAS notions for representing information relevant for agents based on [31].

Agent type	Information	AAS concept
RA	Resource capability	Capability
	Objective function	Submodel element
	Parameters; costs; status information	Property or Range
PA	Set of required processes	Submodel element collection
	Required process	Submodel element
	Static parameters; variables; production status	Property or Range

3.3 Recursive architecture

Taking into account the hierarchical structure of production systems, we rely on a recursive architecture, inspired by holonic systems [35], for RAs, which is complemented by PAs, cp. Figure 2. A RA may consist of an arbitrary number of subordinate RAs. For instance, a plant's RA aggregates its machines' RAs. These subordinate RAs register themselves and their capabilities with the superordinate RA, which serves as a directory facilitator. As an entry point for the RA structure, a single top-level RA has to be specified, which we call the *coordinator*. This coordinator is responsible for initiating the creation of all its subordinate RAs, but it also serves as a DF for PAs. Hence, PAs may request services from this top-level RA, which provides the addresses of the appropriate subordinate RAs. Also, PAs may negotiate the order of resource usage based on their assigned or calculated priority if they need the same resource. All agents within the MAS are connected to the respective asset, and they include a KB, which is generated from the respective DT. Each agent's KB includes the agent's environment model, i. e., its believes, and an objective function, i. e., its desires.

3.4 Environment models for agents

To enable decentralized decision-making, agents require information about their surroundings, the so-called en-

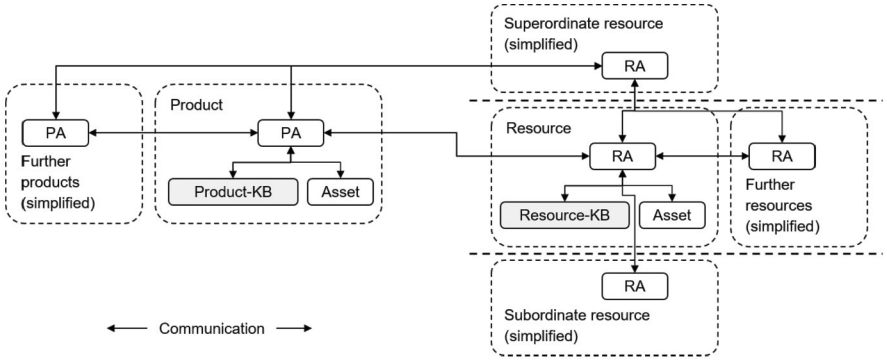

Figure 2: Recursive architecture for the DT-based MAS framework.

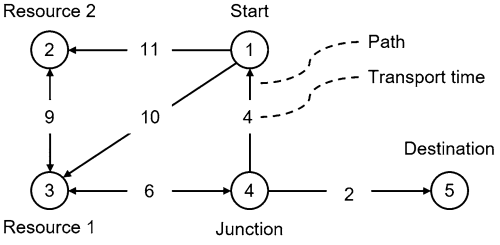

Figure 3: Example for an environment model represented as a directed graph.

vironment model. In the context of this work, the environment model includes an overview of the resources, their capabilities, and their connections via transport resources, which is stored in the form of a transport graph. Since all crucial decisions are made by PAs, the environment models are crucial only for PAs and the RAs that serve as DFs. In contrast, the other RAs require only descriptions of their capabilities. A transport graph is a potentially cyclic and labelled graph, which represents resources as nodes, transport processes as edges, and the costs for transport processes as weights associated to the edges, cp. Figure 3.

For ensuring efficiency, the KBs of agents are limited to the information that is actually relevant for them. There are multiple ways for an agent to receive an environment model. It can be stored in their KB before initialization, they can build it themselves by exploration, or another entity creates and provides it to the agent. While a preconstructed environment model saves time and computation resources during run time, it is static. If agents create their environment themselves by perceiving their surroundings they are completely independent from other entities, but the environment model may be incomplete, and its creation is potentially computationally expensive. To reduce computational cost for the individual agents, they may also exchange information regarding their shared surroundings. This, however, results in an increase in communication required. As a compromise between these options, we rely on an environment model which is centrally created by the coordinator for each agent from the respective DT. This happens when the individual agents are initialized, but before they start up. To represent changes in the layout, e. g., due to degradation, stalled RAs or disruptive employees, the agents KBs are continuously adapted, though, similar to existing approaches [9]. The responsibility for keeping track of changes and initiating KB changes falls to the coordinator, as it has an overview which information and thus which changes are of relevance for which agents. Pragmatically, the coordinator can base this overview of which RA needs which information on its prior interactions with the RAs, which already requested certain information.

3.5 Instantiating individual agents from Digital Twins

To leverage DTs for MASs, we suggest automatically creating the MAS' agents from the respective DTs, cp. Algorithm 1. To do so, we first create the coordinator and the AMS, as all other agents depend on them. The coordinator then uses its own KB, to infer which RAs it has to create for its subordinate resources. Hereby, it relies on the links *RDT_files* to the subordinate DTs specified in the coordinator's DT. These RAs then parse their DTs to initialize their KBs. RAs and PAs read the respectively relevant information, such as capabilities and required processes, from the DTs and store this information in a KB, which can be accessed easily and quickly during runtime. As the DTs include all relevant information like properties and abilities required for decision-making and bidding processes, the agents can create their KBs completely autonomously from

Algorithm 1 Agent initialization

```
1:  procedure AGENTINIT(coordinatorFile, pdtFiles)
2:    initializePadeMas()
3:    initCoordinator(coordinatorFile)
4:    for all rdt in coordinator.rdtFiles do
5:      ra = coordinator.createRaInPade()
6:      ra.parse(rdt)
7:      ra.writeToKb()
8:    for all pdt in pdtFiles do
9:      pa = createPaInPade()
10:     pa.parse(pdt)
11:     pa.writeToKb()
```

the DTs. Subsequently, the coordinator builds a model of the entire production system that revolves around the transport graph. Whenever a product is to be produced, a PA is created, which initializes its KB from a DT that includes relevant static information regarding dimensions, but also priority, destination, and processes required. For this, we assume that a list of DTs for the products is provided. The PA's dynamic status information such as current location and remaining processes required are updated continuously. Subsequently, the PA requests an environment model from the coordinator to be able to start its pathfinding algorithm. The coordinator as a centralized entity keeps track of the entire plants status in the form of the transport graph, which it provides to other agents. It updates this transport graph upon its initialization, but also whenever the plant's layout changes due to engineering modifications or if a resource breaks. These changes are either communicated by the asset's RA, or detected by the AMS. Either way, the coordinator is prompted to update its environment model and communicate the changes to all agents.

3.6 Decision-making process

The decision-making process consists of two core parts, namely the bidding process and the path finding algorithm. While the bidding process is the basis for determining which resources execute the processes a product requires, the path finding algorithm aims to find the shortest path to that next resource. For the bidding process, we rely on the established auction protocols defined by the Foundation for Intelligent Physical Agents (FIPA) [36], cp. Figure 4. Here, the PAs and RAs register with the DF. Then, any PA can request the addresses of agents with the capabilities required from the DF and directly contact these RAs to inquire their availability, these are so-called calls for proposals. Based on the proposals made by the RAs, the PAs can optimize their next step regarding their objective function, in our case simply for time, and decide where to be manufactured. Subsequently, the PAs inform the RA about their decision by accepting or rejecting the RAs proposals.

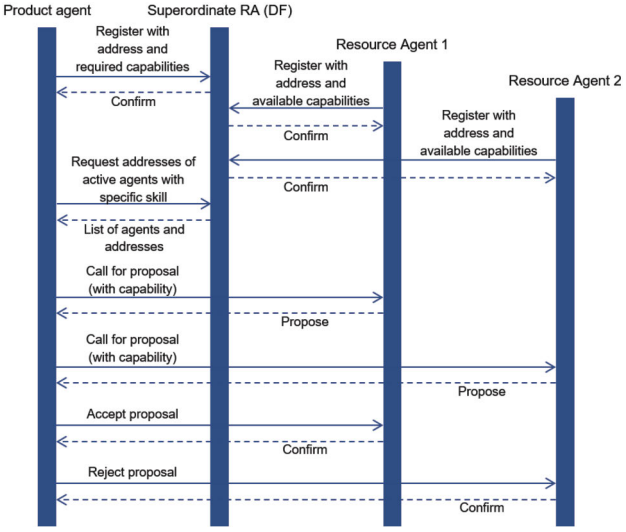

Figure 4: Excerpt of the sequence diagram representing the bidding process between agents.

Upon receiving the RAs' proposals, the PA optimizes its next step, taking into account the offers of the RAs including cycle times and waiting times as well as transport costs, including transport times. Hereby, "shortest" may also denote "the least expensive", depending on what is encoded into the transport graphs labels. For finding the shortest path, we use the established Dijkstra algorithm. Algorithm 2 gives an overview of the entire procedure of PAs negotiating with RAs and optimizing their path. Here, *nextPath* is a PA-specific ordered list of nodes, starting with the product's current location as *nextPath*[0], followed by the locations of the RAs to be visited. The *CycleTime* is the time a resource needs from starting a process for a product until it is ready to execute a process on another product, while the *WaitingTime* is the sum of all currently scheduled processes of a resource. Note that the function *optimizeWithDijkstra* returns exactly one possible shortest path. After having made an informed decision where to be processed further, the PA notifies the respective RAs by accepting or rejecting the RA's proposal, cp. Figure 4.

Algorithm 2 Pathfinding

```
1: procedure PathFinding(network, tasklist, location, destination)
2:   while tasklist and location ≠ destination do
3:     if tasklist then
4:       for all task ∈ tasklist do
5:         for all ra ∈ task.ras do
6:           requestRaOffer(ra, cycleTime, waitingTime)
7:           network.insertNode(virtualNode)
8:           network.insertWeightedEdge(cycleTime)
9:           network.insertWeightedEdge(waitingTime)
10:      possiblePaths = [optimizeWithDijkstra(network,
           location(pa), location(ra)) for ra in task.ras]
11:      nextPath = min(possiblePaths)
12:      confirm(ra(nextPath))
13:      reject(ra) for ra in task.ras if ra ≠ ra(nextPath)
14:      for all node ∈ nextPath[1 :] do
15:        requestTransport(node)
16:      if task.done == True then
17:        tasklist.remove(task)
18:    else if location ≠ destination then
19:      nextPath = optimizeWithDijkstra(network,
           location(pa), destination)
20:      for all node ∈ nextPath do
21:        requestTransport(node)
```

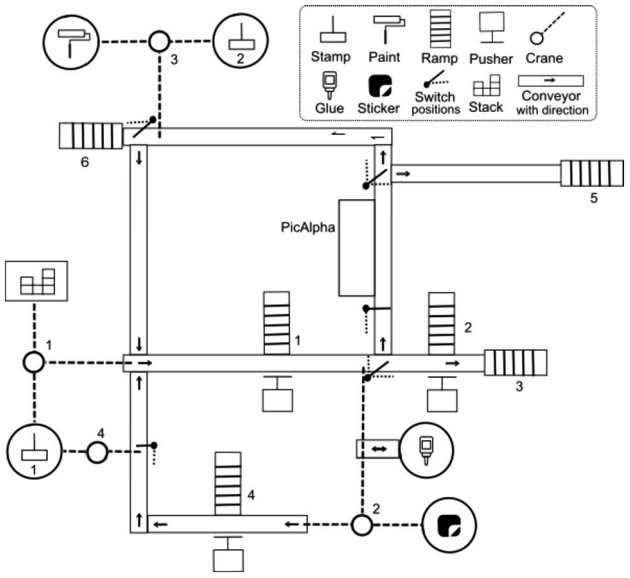

Figure 5: Adapted layout of the extended Pick and Place Unit (xPPU).

4 Implementation and discussion

This section gives an overview of the demonstrator, describes details of the implementation, provides details regarding the approach's applicability through a simulation-based evaluation, and discusses the results.

4.1 Demonstrator

The extended Pick and Place Unit (xPPU) is a demonstrator for industrial plant automation in general and evolving production systems in particular [37]. The basic Pick and Place Unit (PPU) consists of a stack, a crane, and a ramp. This configuration was extended to include conveyors, a PicAlpha module, several ramps, RFID readers, a stamping module, safety modules, switches, and more. However, even the advanced evolution scenarios of the xPPU include still little redundancy in the resources. Specifically, the original layout is close to a static production line, which makes the decision-making process of PAs trivial, as there is only exactly one RA available for each task. To demonstrate the potential of an agent-based approach, we created a more complex scenario that includes redundancies, cp. Figure 5. This extended version of the xPPU includes more conveyors, two more cranes, a second stamping module, a painting module, a resource that can apply stickers, and gluing modules. The resulting decision making processes for PAs are non-trivial and thus the layout is suitable to validate that the architecture works as intended.

We created DTs for all the resources in this extended layout, which include relevant information such as capabilities, but also the resource's location. For an excerpt see Figure 6. For this, we relied on available engineering documents whenever possible, but had to curate the information mostly manually.

4.2 Implementation details

For both MASs and DTs, there are various established and tested frameworks available. Hence, we had to make two essential choices regarding the frameworks to be used for the prototypical implementation.

Established MAS frameworks we considered include JADE [14], its Python counterpart PADE[5] [15], and Pykka. JADE is fully FIPA-compliant, and PADE supports at least Agent Communication Language (ACL)-messages and se-

5 PADE: https://pade.readthedocs.io/en/latest/ last accessed: November 24, 2021.

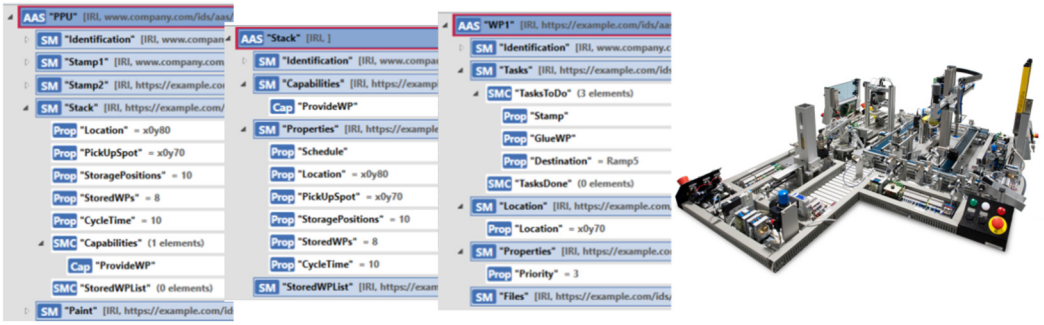

Figure 6: Excerpts of the AASs for the xPPU, its stack and a workpiece.

lected protocols out of the box, while both have to be implemented manually for Pykka. Both JADE and Pykka run on multiple threads while the PADE platform runs on a single asynchronous thread. JADE automatically runs an AMS and a DF, where the AMS creates and destroys agents, and the DF lists all active agents on the platform. PADE also starts up an AMS, which works differently. In PADE, each agent has a table with all active agents and their communication addresses. If a new agent starts up, it has to get registered by the AMS which then tells all active agents about the new one. Both PADE's and JADE's AMSs try to restart agents if they fail. Additionally, PADE supports serializing objects with the Python library pickle and sending them as ACL-message content. PADE was chosen for its AMS, Python usage, and its ability to send serialized objects. An included AMS reduces the implementation effort and the version used by PADE fulfils the tasks assigned to the AMS in the architecture, such as agent registration and restarting. Furthermore, we relied on Python libraries for XML-file reading and editing,[6] graph creation and path finding using Dijkstra's algorithm (specifically NetworkX' *single_source_dijkstra* function),[7] graph plotting,[8] and object serialization.[9] Further, PADE allowed us to send the serialized objects as the content of ACL-messages. In the context of our implementation, these objects are transport graphs and location dictionaries. If necessary, one could also send entire AAS-files, which can be serialized as XML files [22].

Regarding the DT format, the approach presented requires passive or reactive DTs only, which are able to store information in a standardized format and structure. These DTs, which serve as basis for the agents KB should be easily accessibly and editable. C2PS focuses on the architecture and decision-making process of a DT and does not provide details for how to store the data. Additionally, only a reference architecture is provided, which means that the framework would have to be re-implemented. Both the DTDL and AAS specification provide a structure and format for storing the DT's information. Even though both can be extended with self-defined properties and models, the AAS metamodel provides more details relevant for the production context out of the box. Furthermore, various editors are available for the AAS. For initially creating the AASs for the xPPU's resources, we used the AASX Package Explorer.[10] An excerpt of the xPPU's stamp's AAS, including its stamping capability, is depicted in Figure 7.

There are advanced tools available for editing AAS files, e. g., the PyI40AAS.[11] PyI40AAS is a module that enables modeling of AASs as Python objects, (de-)serialization of AAS objects from and to JSON and XML, reading and writing of AASX package files, and compliance checking of AAS files. However, we only have to make minor modifications to the agents' AASs files, e. g., marking required processes as executed. Hence, for simplicity, we chose Python's etree package for interacting with the AAS files.

6 Python Software Foundation's etree module: https://docs.python.org/3/library/xml.etree.elementtree.html last accessed: November 24, 2021.

7 NetworkX: https://pypi.org/project/networkx/ last accessed: November 24, 2021.

8 Matplotlib's pyplot module: https://matplotlib.org/stable/api/pyplot_summary.html last accessed: November 24, 2021.

9 Python Software Foundation's pickle module: https://docs.python.org/3/library/pickle.html last accessed: November 24, 2021.

10 https://github.com/admin-shell-io/aasx-package-explorer last accessed: November 24, 2021.

11 https://git.rwth-aachen.de/acplt/pyi40aas last accessed: November 24, 2021.

Figure 7: Exemplary capability modeled in the AAS.

4.3 Evaluation

For evaluating the framework developed, we ran a quantitative performance assessment and assessed it qualitatively regarding the Cyber-Physical Production Systems (CPPSs) requirements defined by Ribeiro and Hochwallner [38].

For evaluating performance, we relied on the use case described in Section 4.1. The resulting MAS consisted of 8 PAs and 19 RAs. So far, the initialization of the agents was unexpectedly slow with approximately 10 s per agent. However, these initializations do not occur often and are not time critical. Spikes may be due to inefficient exchange of environment models and shall be optimized for performance in future work. Even without performance optimization, the agents' mean reaction times, cp. Figure 8, seem acceptable compared to the typical production process and transport times. Also, PADE has been proven to be able to perform well, even for large and potentially industrial MASs with up to 400 agents being run in parallel [15], allowing the conclusion that the current proof-of-concept framework can be optimized appropriately. First, the number of threads increases steadily as threads are started for the individual agents. Later, the number of threads varies as the agents start and stop further threads as required for some asynchronous tasks like the PAs' decision-making algorithms or the RAs' schedule tracking.

Ribeiro and Hochwallner [38] propose assessing CPPSs regarding adaptability, convertibility, and integrability. Their scale ranges from local autonomy and basic protocols (level 1) up to cyber-physical autonomy and the ability to dynamically change the system's structure (level 5). According to their analysis, current production systems reach level 2 while some research projects can reach level 4. Regarding adaptability, the agents make decisions based on their environment models, which are created from DTs and are updated in case the production system changes. In the current implementation, these updates are initiated by the coordinator and the focus was not put on perceiving abilities of agents so that they could update their environment models autonomously. Also, the framework does not yet provide learning capabilities. It may be extended for both aspects, though. Optimal adaption is realized for certain production aspects, as the PAs can implement several path finding algorithms. Resilience is supported by enabling restarts of agents and recalculation of paths in case of broken resources. However, the physical layout of the production resources can not be changed by the system itself. Since we assume that the PAs only request single processes, the framework is not built to support collaboration of RAs within the same pro-

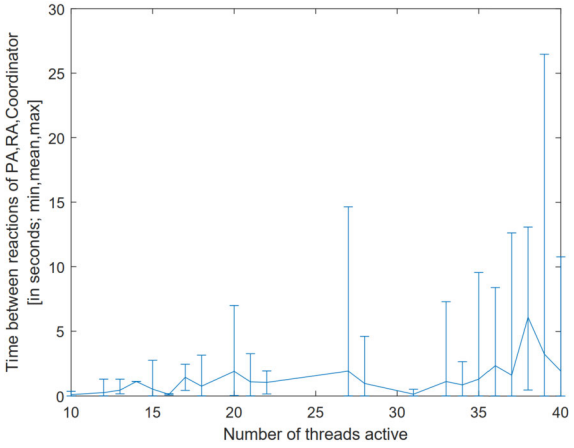

Figure 8: Reaction times of the agents depending on the number of threads. Time between reactions = time that passes between execution of the line of code that sends a message and the execution of the line of code that triggers the respective response.

cess, but only in the overall production process. Convertibility is in principle supported, as we assume the CPPS to have a modular physical structure. Additionally, agents, especially when based on modular KBs, are intrinsically quick to adjust to changing environments. Integrability is addressed by relying on the AAS as a basis for standardized KBs. This a priori standardization, especially when combined with the ECLASS standard, ensures semantic interoperability between agents. However, it is based on the assumption, that all stakeholders involved adhere to the standard chosen. The aspects diagnosability, safety, and security have not been explicitly considered, but they can be addressed by relying on work developed within existing agent frameworks [2, 10, 13]. In summary, this contribution focuses on leveraging DTs for MASs and does not explicitly address all the requirements for CPPSs defined by Ribeiro and Hochwallner [38]. However, it does so implicitly, as it relies heavily on the agent paradigm.

4.4 Discussion

Intelligent agents in a MAS in the production context require various pieces of information to make decisions autonomously. DTs in general, and the AAS in particular, have been shown to be able to provide this information for both RAs and PAs, cp. Table 1 *(RQ1)*. However, such a priori approaches intrinsically require that all stakeholders involved have the same understanding of the underlying metamodel. Different understandings would reflect not only in the DTs' structure, which could theoretically be identified automatically, but also in the way information is represented. Due to the complexity of the AAS metamodel, freedom with regard to its use, e. g., the way submodels are nested, and the fact that it is still under development, using it correctly is still challenging. The same challenges apply to ensuring semantic interoperability via ECLASS. Lastly, it is cumbersome to create DTs manually, even though there are tools available, such as the AASX Package Explorer.

Since the information relevant for RAs and PAs can be captured using DTs, these agents can be initialized from their DTs *(RQ2)*. Here, it is of importance that the DTs' structure represents the production system's modular architecture and that hierarchical relations are appropriately represented. After specifying the top-level DT as an entry point, the approach can automatically identify all subordinate resources and create RAs for them. PAs on the other hand can be added on the fly, which corresponds to the paradigm of order controlled production [2]. Automatically extracting the information from the DTs is enabled by the AAS specification [22], which also clearly defines the serialization. Potentially, information can be fed back into the DTs analogously. The prototype showed promising results, even though performance would need to be improved for industrial applications.

Regarding the operation of MAS *(RQ3)*, DTs can be integrated flawlessly with existing MAS frameworks and standards, e. g., PADE. This allows engineers to build on established technologies, but leverage DTs to overcome previous limitations. Even though there are still challenges in applying the AAS as a common metamodel and ECLASS for avoiding ambiguities, this standardization would pave the way to unambiguous communication and thus greatly benefits MAS too. This potential has been demonstrated using a simulation of a scaled-up version of the lab-scale demonstrator xPPU. Future work will evaluate how this system performs when connected to the physical system, for different plants and also in an industrial setting. There, further challenges shall be researched, for instance regarding timeliness. Finally, it has to be noted that the system developed aims to create RAs and PAs, which are not required to meet hard realtime requirements. For the fieldlevel, a more hierarchical structure with dominant and submissive agents instead of negotiating ones would be more appropriate. Even though we believe that such realtime-capable architectures would also benefit from leveraging DTs, it still has to be investigated how the differences in DTs for fieldlevel components compared to higher-level modules would influence the approach's applicability.

5 Summary and outlook

Even though MASs have the potential to realize flexible and adaptable production systems, their application is still limited by ambiguities in the agents' communication and the effort for creating the individual agents' KBs. The approach presented leverages DTs in the form of AASs to address this challenge. As a basis, we presented an overview of information required by the different agents and showed that it can be represented using the AAS metamodel *(RQ1)*. Using DTs with this information as an input, we showed how a MAS can be created and how the individual RAs and PAs can be automatically initialized from their respective DTs *(RQ2)*. As a proof of concept, we presented how DTs can be leveraged to set up a MAS in PADE for an extended version of the demonstrator xPPU *(RQ3)*. The approach was evaluated using a simulation of this extended PPU. In summary, this approach shows the potential of leveraging DTs for MASs, which will hopefully pave

the way towards making MASs mainstream and eventually easily applicable for industry.

However, there are still several open research challenges to be addressed in future work. The effort for creating DTs, e. g., with the AASX Package Explorer, is problematic, as it will prevent acceptance. Hence, approaches should be developed that would ideally automatically create DTs, for instance from existing engineering documents, or at least support engineers in creating the DTs. Also, we assumed that products are described in terms of required production processes. To lift this assumption, an automated matching of product features to production processes would be required. Even though this is challenging, it would greatly benefit the combination of the perspectives product and resource, and their respective agents. Lastly, it is essential that the DT accurately represents the asset, especially if it is used as an input for creating an agent's KB. Only if the KB is up to date can the agent make appropriate decisions, if necessary also using a more complex objective function, which also mirrors criteria such financial costs and quality instead of time only. For this, the DT has to be coupled to the asset and the two must be synchronized continuously. Since the agents include up-to-date environment models, they could also be leveraged to feed updates back into their DTs. Here, especially the thought of peer-supervision among agents is promising, i. e., agents notify the coordinator of unexpected behavior of their peers. The coordinator can then use this information to provide updated environment models to the agents affected. Hereby, it is essential to avoid and possibly resolve inconsistencies between the asset and the DT, which still imposes a challenge, especially because timeliness is of importance. If that is ensured, the DT, e. g., in the form of the reactive AAS, may also serve as an abstraction layer between the agent and the asset, thus further facilitating the set-up of the MAS.

Funding: We kindly thank the German Federal Ministry of Education and Research (BMBF) for funding the project DAVID (01IS18075F).

References

1. H. ElMaraghy, A. AlGeddawy and A. Azab, "Change in Manufacturing – Research and Industrial Challenges," in *Enabling Manufacturing Competitiveness and Economic Sustainability*, pp. 2–9, 2011.
2. B. Vogel-Heuser, M. Seitz, L. A. Cruz Salazar, F. Gehlhoff, A. Dogan and A. Fay, "Multi-agent systems to enable Industry 4.0," *at – Automatisierungstechnik*, vol. 68, no. 6, pp. 445–458, 6 2020.
3. B. Vogel-Heuser, J. Lee and P. Leitão, "Agents enabling cyber-physical production systems," *at – Automatisierungstechnik*, vol. 63, no. 10, pp. 777–789, 2015.
4. M. Wooldridge, "Intelligent agents: The key concepts," in *ECCAI Advanced Course on Artificial Intelligence*. Springer, 2001, pp. 3–43.
5. VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik, "VDI/VDE 2653-1: Multi-agent systems in industrial automation – Fundamentals," 2018. [Online]. Available: https://www.vdi.de/richtlinien/details/vdivde-2653-blatt-1-agentensysteme-in-der-automatisierungstechnik-grundlagen.
6. L. A. Cruz Salazar, D. Ryashentseva, A. Lüder and B. Vogel-Heuser, "Cyber-physical production systems architecture based on multi-agent's design pattern – comparison of selected approaches mapping four agent patterns," *International Journal of Advanced Manufacturing Technology*, vol. 105, no. 9, pp. 4005–4034, 2019.
7. VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik, "VDI/VDE 2653-4: Multi-agent systems in industrial automation – Selected patterns for field level control and energy systems," 2021. [Online]. Available: https://www.vdi.de/richtlinien/details/vdivde-2653-blatt-4-multi-agent-systems-in-industrial-automation-selected-patterns-for-field-level-control-and-energy-systems.
8. P. Verstraete, B. Saint Germain, K. Hadeli, P. Valckenaers and H. Van Brussel, "On applying the PROSA reference architecture in multi-agent manufacturing control applications," in *Proceedings of the Multi-agent Systems and Software Architecture Special Track at Net. ObjectDays*, pp. 31–47, 2006.
9. I. Kovalenko, "Intelligent product agents for multi-agent control of manufacturing systems," Ph.D. dissertation, University of Michigan, 2020. [Online]. Available: https://deepblue.lib.umich.edu/handle/2027.42/162893.
10. J. Peschke, A. Lüder and H. Kühnle, "The PABADIS'PROMISE architecture," *IEEE Procedings*, vol. 1, pp. 491–496, 2005.
11. A. M. Farid and L. Ribeiro, "An Axiomatic Design of a Multiagent Reconfigurable Mechatronic System Architecture," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 5, pp. 1142–1155, 2015.
12. P. Leitão and F. Restivo, "ADACOR: A holonic architecture for agile and adaptive manufacturing control," *Computers in Industry*, vol. 57, no. 2, pp. 121–130, 2006.
13. S. Rehberger, L. Spreiter and B. Vogel-Heuser, "An agent-based approach for dependable planning of production sequences in automated production systems," *at – Automatisierungstechnik*, vol. 65, no. 11, pp. 766–778, 2017.
14. F. Bellifemine, A. Poggi and G. Rimassa, "JADE – A FIPA-compliant agent framework," in *Proceedings of PAAM*, pp. 97–108, 1999.
15. L. S. Melo, R. F. Sampaio, R. P. S. Leão, G. C. Barroso and J. R. Bezerra, "Python-based multi-agent platform for application on power grids," *International Transactions on Electrical Energy Systems*, vol. 29, no. 6, pp. 1–14, 2019.
16. T. Wagner, "Agentenunterstütztes Engineering von Automatisierungsanlagen," in *VDI Berichte*, no. 1883, pp. 559–567, 2005.
17. R. N. Bolton, J. R. McColl-Kennedy, L. Cheung, A. Gallan, C. Orsingher, L. Witell and M. Zaki, "Customer experience challenges: bringing together digital, physical and social realms," *Journal of Service Management*, 2018.

18. F. Tao, H. Zhang, A. Liu and A. Y. Nee, "Digital Twin in Industry: State-of-the-Art," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, pp. 2405–2415, 2019.
19. E. Bayrhammer, F. Ocker, E. Trunzer, M. Eisenträger, C. Bornstein, A. Strahilov, H. Avgoustinos, S. Himstedt, S. Adler and B. Vogel-Heuser, "Verteilte Digitale Zwillinge – der Stand der Technik, Anwendungsfälle und Zielstellung," in *VDI-Kongress AUTOMATION*, 2020.
20. Y. Lu, C. Liu, I. Kevin, K. Wang, H. Huang and X. Xu, "Digital twin-driven smart manufacturing: Connotation, reference model, applications and research issues," *Robotics and Computer-Integrated Manufacturing*, vol. 61, 2020.
21. J. Moyne, Y. Qamsane, E. C. Balta, I. Kovalenko, J. Faris, K. Barton and D. M. Tilbury, "A Requirements Driven Digital Twin Framework: Specification and Opportunities," *IEEE Access*, vol. 8, 2020.
22. Plattform Industrie 4.0, "Details of the Asset Administration Shell – Part 1 (Version 3.0RC01)," 2020. [Online]. Available: https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V3.html.
23. B. Boss, S. Malakuti, S.-W. Lin and T. Usländer, "Digital Twin and Asset Administration Shell Concepts and Application in the Industrial Internet and Industrie 4.0," 2020. [Online]. Available: https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/Digital-Twin-and-Asset-Administration-Shell-Concepts.html.
24. Plattform Industrie 4.0, "Verwaltungsschale in der Praxis – Wie definiere ich Teilmodelle, beispielhafte Teilmodelle und Interaktion zwischen Verwaltungsschalen," 2020. [Online]. Available: https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/2020-verwaltungsschale-in-der-praxis.html.
25. A. Belyaev and C. Diedrich, "Aktive Verwaltungsschale von I4.0-Komponenten – Erscheinungsformen von Verwaltungsschalen," in *Leitkongress der Mess-und Automatisierungstechnik*, 2019.
26. K. M. Alam and A. El Saddik, "C2PS: A digital twin architecture reference model for the cloud-based cyber-physical systems," *IEEE Access*, vol. 5, pp. 2050–2062, 2017.
27. W. Zhang, G. Wang, Y. Yan, H. Chu, J. Wang and Z. Cao, "Intelligent test of spacecraft based on digital twin and multi-agent systems," *Computer Integrated Manufacturing Systems*, vol. 27, no. 1, pp. 16–33, 2021.
28. V. Laryukhin, P. Skobelev, O. Lakhin, S. Grachev, V. Yalovenko and O. Yalovenko, "The multi-agent approach for developing a cyber-physical system for managing precise farms with digital twins of plants," *Cybernetics and Physics*, vol. 8, no. 4, pp. 257–261, 2019.
29. A. Niati, C. Selma, D. Tamzalit, H. Bruneliere, N. Mebarki and O. Cardin, "Towards a digital twin for cyber-physical production systems: a multi-paradigm modeling approach in the postal industry," in *International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2020.
30. X. Zheng, F. Psarommatis, P. Petrali, C. Turrin, J. Lu and D. Kiritsis, "A quality-oriented digital twin modelling method for manufacturing processes based on a multi-agent architecture," *Procedia Manufacturing*, vol. 51, pp. 309–315, 2020.
31. F. Ocker, C. Urban, B. Vogel-Heuser and C. Diedrich, "Leveraging the Asset Administration Shell for Agent-Based Production Systems," in *IFAC Symposium on Information Control Problems in Manufacturing*. Elsevier, 2021.
32. F. Ocker, I. Kovalenko, K. Barton, D. Tilbury and B. Vogel-Heuser, "A Framework for Automatic Initialization of Multi-Agent Production Systems Using Semantic Web Technologies," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4330–4337, 2019.
33. F. Ocker, B. Vogel-Heuser and J. Fischer, "Towards Providing Feasibility Feedback in Intralogistics Using a Knowledge Graph," in *International Conference on Industrial Informatics*, 2021.
34. S. Bougouffa, K. Meszmer, S. Cha, E. Trunzer and B. Vogel-Heuser, "Industry 4.0 interface for dynamic reconfiguration of an open lab size automated production system to allow remote community experiments," in *International Conference on Industrial Engineering and Engineering Management*, 2018.
35. A. Giret and V. Botti, "Holons and agents," *Journal of Intelligent Manufacturing*, vol. 15, no. 5, pp. 645–659, 2004.
36. Foundation for Intelligent Physical Agents, "FIPA Agent Communication Language," 1997. [Online]. Available: http://www.fipa.org/specs/fipa00018/OC00018.pdf.
37. B. Vogel-Heuser, S. Bougouffa and M. Sollfrank, "Researching Evolution in Industrial Plant Automation: Scenarios and Documentation of the extended Pick and Place Unit," Institute of Automation and Information Systems, Technical University of Munich, Tech. Rep., 2018. [Online]. Available: https://mediatum.ub.tum.de/node?id=1468863.
38. L. Ribeiro and M. Hochwallner, "On the design complexity of cyberphysical production systems," *Complexity*, 2018.

Bionotes

Birgit Vogel-Heuser
Institute of Automation and Information Systems, Department of Mechanical Engineering, TUM School of Engineering and Design, Core Member of MDSI and Member of MIRMI, Technical University of Munich, Munich, Germany
Prof. Dr.-Ing. Birgit Vogel-Heuser is also Core Member of MDSI and Member of MIRMI
vogel-heuser@tum.de

Birgit Vogel-Heuser, Prof. Dr.-Ing., is a full professor and director of the Institute of Automation and Information Systems at the Technical University of Munich. Her main research interests are systems engineering, software engineering, and modeling of distributed and reliable embedded systems. She is core member of TUM's MDSI, member of TUM's MIRMI, member of the German Academy of Science and Engineering, chair of the VDI/VDE working group on industrial agents, vice chair of the IFAC TC 3.1 computers in control, and was coordinator of the Collaborative Research Centre (CRC) 768: Managing cycles in innovation processes – integrated development of product-service systems based on technical products.

Felix Ocker
Institute of Automation and Information Systems, Department of Mechanical Engineering, TUM School of Engineering and Design, Technical University of Munich, Munich, Germany
felix.ocker@tum.de

Felix Ocker, M. Sc., is a graduate research assistant and Ph. D. student with the Institute of Automation and Information Systems at the Technical University of Munich. His research focuses on knowledge formalization and inconsistency management in interdisciplinary engineering.

Tobias Scheuer
Institute of Automation and Information Systems, Department of Mechanical Engineering, TUM School of Engineering and Design, Technical University of Munich, Munich, Germany
tobias.scheuer@tum.de

Tobias Scheuer, B. Sc., is an undergraduate research assistant and master student in Mechatronics and Robotics at the Technical University of Munich. His research interests lie in automation, multi-agent systems, and digital twins.