# Towards developing Digital Twin enabled Multi-Agent Systems*

Stefano Mariani[1], Marco Picone[1], and Alessandro Ricci[2]

[1] Department of Sciences and Methods of Engineering, University of Modena and Reggio Emilia, Italy name.surname@unimore.it
[2] Department of Computer Science and Engineering, University of Bologna, Italy a.ricci@unibo.it

**Abstract.** The Multi-Agent Systems (MASs) literature provides abstractions, techniques, and development platforms to design and implement the *virtual environment* within which agents operate. However, coupling such an environment with a *physical* counterpart is still cumbersome, as existing approaches deal with the issue in an ad-hoc way, without general purpose abstractions and methods. Recently, a new paradigm could complement the agent-oriented one to deal with digitalisation of physical environments in a more principled and interoperable way: the Digital Twin (DT). In this paper, we propose a first principled integration between MAS and DTs for MAS environment engineering.

**Keywords:** Digital Twin · Multi-agent System · WDLT · JaCaMo.

## 1 Introduction

Multi-Agent Systems (MAS) are the premiere source of abstractions and methods (and programming and execution platform as well) to model and engineer *complex* systems [7]. Examples include Cyber-Physical Systems (CPS) [5], e.g. the monitoring and control software of a manufacturing factory, where agents collect measurements from machinery and equipment (i.e. their digital representations) to support human supervision and decision making; Web of Things deployments [2], e.g. the software controlling energy consumption of smart appliances in a smart building like an hotel, where different agents are in charge of negotiating the best settings to find the optimal trade-off against competing interests (e.g. management's cost saving policies and guests' comfort).

The MAS literature provides plenty of agent models and development (and execution) platforms, ranging from simple *reactive* agents mostly used for simulation [25, 15], to pro-active *cognitive* agent architectures meant to autonomously carry out sophisticated reasoning [8, 21]. There are also models and methods to engineer the environment that agents must interact with to carry out their duties, such as the A&A meta-model [17] providing *artefacts* as the first-class abstraction meant to digitally represent both physical resources and legacy software

* Work partially supported by Italian PRIN "Fluidware" (N. 2017KRC7KT).

(e.g. databases or external services). However, the nuts and bolts of connecting a digital representation to its physical counterpart (e.g. an individual sensor, a manufacturing equipment, or even a whole production line), and the implementation of the process that keeps the two *aligned* at all times are not well engineered. Indeed, often the focus is on the *interface* between the agent and the artefact (or whatever other abstraction is provided), not between the artefact and the physical *thing*. This forces programmers to "re-invent the wheel" for every new development, or hides potentially reusable designs in each team or organisation own implementations, leading to fragmentation.

A solution could come from *Digital Twins* (DTs) [14], that is, digital representations of an (physical) entity of interest (e.g. object, location, person, process) *continuously reflecting* its state and behaviour in a software object, meant to provide services to other software entities (e.g. business applications) [22]. Amongst the many applications of the concept [27], that of exposing a *uniform and interoperable* digital layer to applications and services, tightly coupled with the physical world but hiding to such applications the heterogeneity and complexity of managing resources and processes, is relevant for MAS engineering.

Accordingly, in this paper we propose DTs as a *complement* to existing models and methods for MAS environment engineering, with the goals of (i) achieving a principled way to couple digital representations of entities to their physical counterparts, and (ii) decouple MAS environment engineering methods from the intricacies and peculiarities of accessing to and interacting with physical devices. We argue, in fact, that it is conceptually wrong, and technically inconvenient, to model a DT, or a CPS component, as an agent. For the former, modelling DTs as agents would clash with the definitions we adopt (see Section 3.1); for the latter, DTs are better candidates to model them. MAS designers would gain tangible benefits in terms of (i) *separation of concerns*, as they can engineer their solution in terms of MAS abstractions without "polluting" them with devices or protocol-specific technicalities, and (ii) *independent evolution*, as once the DTs interface to the MAS is established, developers of the MAS functionalities and those managing the physical layer can evolve their implementations separately.

## 2 State of the art

The vision we aim to realise with this paper is aligned with the view fostered in [12], where agents and DTs are seen as complementary abstractions whose principled integration can bring benefits to two tasks, mostly: (i) engineering the MAS environment, and (ii) orchestrating and coordinating (e.g. dynamically compose) agents and DTs' offered services. In particular, we exploit the kind of *separation of concerns* therein defined, where DTs are meant to operate (i.e. perceive, act) within the boundaries set by the *local* context of their associated physical twin, whereas agents are meant to pursue the application goals in the *global* context of all the resources and services available to the whole MAS. Figure 1 in Section 3 depicts our envisioned architecture aligned with this view.

There are a few works in the literature about exploiting agents and DTs synergistically in the perspective described above, that is, where DTs take care of interacting with the environment on behalf of agents, and agents use and orchestrate DTs for achieving their goals. For instance, in [3] DTs model environment resources so as to support the agents' decision making, while agents gather knowledge from multiple DTs to achieve their goals. In [16] a specific instance of the concept of DT, called "Asset Administration Shell" (AAS), enables agents' operations on a physical production system, by mediating access to all the different physical devices. In [10] DTs are used in a manufacturing CPS to better manage communication of data from the physical devices to the MAS monitoring and controlling the system, for instance by performing protocol translation, buffering, etc. Even if restricted to communication issues, DTs actually encapsulate the resources in the MAS environment (the manufacturing CPS). Finally, in [28] a distributed simulation platform is shaped around DTs: each DT simulates a specific asset or set of assets, and agents orchestrate such DTs to dynamically compose them in a single coherent simulation. In a sense, also here DTs encapsulate a portion of the environment, although in this case such environment is purely simulated.

However, in all the aforementioned works, either agents directly interact with DTs [16, 10, 28], or the concept of DT is directly implemented with the abstractions (and techniques) made available by the MAS—in the case of [3], as a CArtAgO artefact [24]. In next section, we propose an integration architecture complementing, not replacing, MAS environment abstractions with DTs.

Besides these research works, there are others that do not explicitly mention DTs but nevertheless aim at controlling CPSs with a MAS, while pursuing a kind of separation of concerns similar to ours. For instance, authors of [26] recognise that the agent is responsible for the high-level control functions, while the physical asset's "controller" ensures the execution of the agent's high-level decision. Furthermore, they advocate the added value brought by the concept of AAS [16] as a way to provide a standardized description of the asset information. In turn, this helps creating the agent's local knowledge in a standard way and thus ensures interoperability. However, such AAS is mostly a data repository, and the physical controller is not further abstracted away. In [4], *resource access* is identified as a common functionality provided by MAS when applied to CPS. In fact, the RAMI reference architecture adopted in the paper deploys agents mostly everywhere, there included the "asset" level. In this paper, we advocate that (and motivate why) DTs should be adopted instead. In [19] there is only Java as the abstraction layer towards the physical system. In [9] "agentification" is heavily used, that is, wrapping of services and resources within an agent, and no further abstraction is provided at the border with the physical layer.

Finally, Multi-Agent Robot Systems (MARS) could be considered as a special case of CPSs, hence efforts to integrate MAS control in multi-robot systems should be taken into account. In MARS architectures, intelligence, proactivity, and social aspects are usually located within agents at the application layer, whereas handling of all the hardware robotic devices and providing functional-

ity for robotics algorithms is responsibility of the lower layers. However, in [6], authors argue for a different separation of concerns, where agents are also used in lower layers as they provide better abstractions for the intelligence required to perform some functional tasks. At the same time, yet, they recognise that there are still components for which the agent abstraction is "just too much", but fail to provide an alternative abstraction besides Robot Operating Systems (ROS) nodes. In this paper, we argue and motivate why DTs could be better candidates for this. In [13] a MARS architecture is proposed where cognitive and operative layers implement separation of concerns between agents and robotic hardware, but where between the agent runtime (JADE) and ROS there is pure Java, and no further abstraction layer. The same happens in [11].

## 3 Integration architecture

First, we propose a *conceptual* architecture not tied to any particular implementation platform, but only to a MAS meta-model (i.e. A&A [17]), in Section 3.1. Then, we follow-up with a *technical* instantiation of such a conceptual architecture with specific technologies (i.e. JaCaMo [1] and the WLDT library [20]), in Section 3.2. The former shows how the different abstractions provided by DTs and MAS fit together in a coherent paradigm for MAS environment engineering—and, engineering of any CPS. The latter clarifies how such a conceptual framework can be realised with current technologies. WLDT is a Java framework providing highly modular and re-usable code to create and maintain DTs of physical world entities (https://github.com/wldt).

### 3.1 Conceptual

Our conceptual integration architecture is depicted in Figure 1. The definition of terms "agent" and "artefact" that we adopt in this paper is taken from the A&A meta-model [17]. The definition of DT is mostly taken from [14], that tries to sort out the many different definitions already existing for DTs in a coherent one. However, we also adopt the systemic view fostered in [22] about the modelling of an ecosystem of DTs semantically interlinked.

The lowest layer is the physical world, where all the objects, resources, devices, people, processes, and every other entity of interest in a given CPS, that is not conceptually suitable to be modeled as an agent, resides. This is a highly heterogeneous world, where virtually every entity has its own access protocol, measurable properties, functionalities, behaviours, etc. With the purpose of making such substrate more homogeneous (e.g. in terms of network access protocols) a DT layer is placed on top, shielding applications from the technical intricacies of the CPS. DTs are, in fact, perfectly suited to encapsulate physical resources and make them accessible to applications. This two layers compose the CPS layer, as the part of the system strictly intertwined with the physical world.

Above them, the MAS layer begins, composed by two sub-layers. The lower one, closer to DTs, is the MAS environment layer, where everything that is

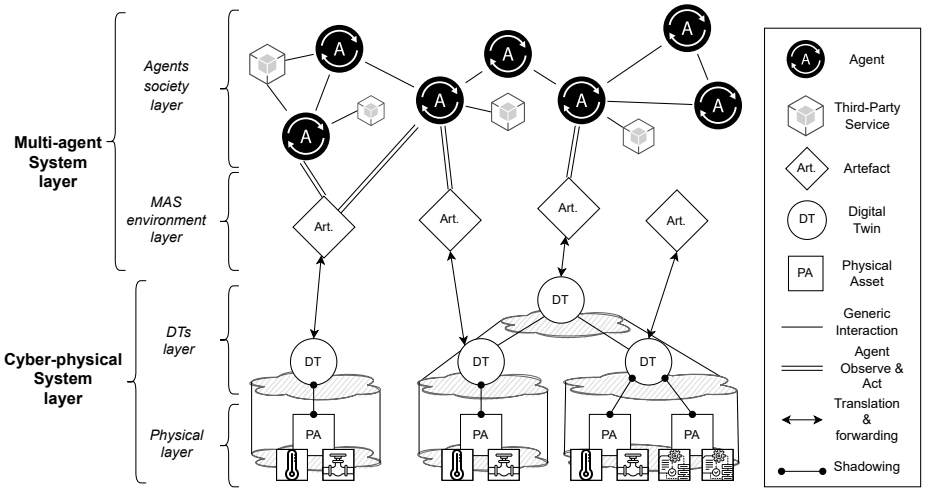


**Fig. 1.** Conceptual integration architecture. DTs complement artefacts in mediating agents' access to physical resources, by shielding artefacts (and agents in turn) from the heterogeneity of communication protocols, data exchange formats, etc., while providing additional services (e.g. fault tolerance, simulation, prediction, etc.).

not an agent is represented. In this paper, non-agent entities are represented as *artefacts* according to the A&A meta-model [17], that is the most principled solution to date [23]. An artefact represents any environmental resource (physical or virtual, such as a database or external service) in terms of admissible actions and available perceptions, perfectly matching most of agent models from reactive to cognitive ones—where an agent is usually defined as an autonomous entity situated in an environment that it can perceive through sensors and act upon through actuators [25]. However, any other environment engineering abstraction would be fine to adopt, as long as it brings the level of abstraction and the programming paradigm closer to the agent-oriented one.

Now, it should be already clear why we crossed the line between CPS and MAS here, at the frontier between DTs and artefacts: the former still promote a development paradigm centred around the physical twin properties and functions – closer to devices –, whereas the latter abstracts them away into perceptions and actions—closer to the agents. In other words, here is where most of the separation of concerns happen.

The highest layer of our conceptual architecture is thus the agents one, where multiple agents cooperate towards the system goals. Such agents actually form a (more or less structured) *society*, that is, a population of agents with roles and missions to accomplish, competing or collaborating within the rules (or "laws") set by the system designer or enforced by some institutional entity that oversees the society as a whole. For instance, in a MAS deployed to control automation of a manufacturing factory, multiple agents can be deployed with different re-

sponsibilities, but is likely that they need to interact in a well structured way to achieve complex system goals (e.g. automate assembly, pick up, and packaging of a product). Structuring the inter-dependencies between these responsibilities can be done via the abstraction of a society, where each agent has a certain role (e.g. the assembler, the collector) and commits to meet others' expectations regarding a specific task or deliverable (e.g. assemble the product correctly, pick up the right parts). Here, agents exploit artefacts as extensions or augmentations of their innate capabilities, or as the mediators of interaction with the resources in their environment—while still reasoning in terms of actions and perceptions. At this level, agents may be completely unaware that artefacts are actually encapsulating DTs, in the same way as DTs may be unaware that their services are being exposed as actions and perceptions to agents. This is the whole point of the principle of separation of concerns and independent evolution brought forward in this paper: they don't need to, and probably don't want to. Agents (as well as their developers) want to think in terms of actions and perceptions on artefacts, whereas DTs (and their developers) need to deal with the physical world technicalities, and do not want to be casted into any specific development framework or mindset dictated by higher layers of the architecture.

However, at the interface between artefacts and DTs, there needs to be a way to "connect" an artefact to a DT and viceversa, while still maintaining loose coupling. How this can be achieved in practice is detailed in next section.

### 3.2 Technical

Figure 2 depicts the technical integration architecture we propose in this paper, as a practical design of the conceptual one described in previous section. As such, it is a zoom-in of the frontier between the MAS environment layer and the DTs layer of Figure 1. The main components of the integration layer are: the `DTDescriptor` and `WLDTDiscoveryService`, that are included in the WLDT library; the `DTWorkspace` and `DTDiscoveryArtefact`, that are newly introduced as part of our technical integration design.

The `DTDescriptor` is a complete description of a given DT provided by the WLDT library: (i) the list of *properties* available for inspection (name-datatype pairs), (ii) the list of *actions* than can be requested to the DT (name, input and output parameters as name-datatype pairs), (iii) the list of *behaviours* that the DT can carry out (name, input parameters, stop condition), (iv) the list of *relationships* the DT has with other DTs (kind, target DT unique identifier), (v) as well as all the metadata needed to interact with the DT—such as an address where to push actions and pull data, the supported protocol(s) (e.g. websocket vs. plain REST CRUD operations), the supported representation format(s) (e.g. JSON, YAML), and any other information needed by external components to directly interact with the DT. Such a descriptor is published by the `WLDTDiscoveryService` to a well known address as soon as a DT is created and bound to its physical twin by the WLDT platform. External components can query it for a list of available DTs, and get the descriptor of any of them.

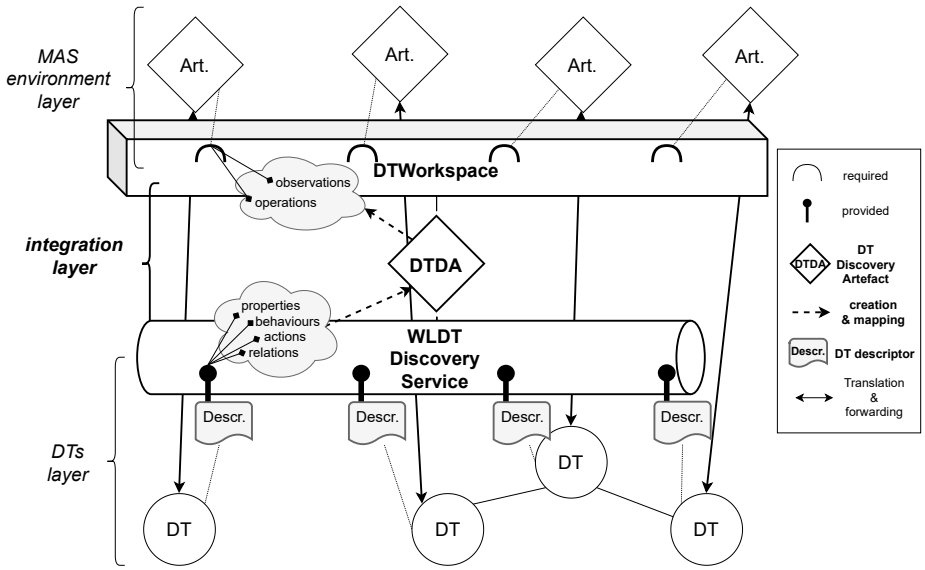


**Fig. 2.** Technical integration architecture. The `DTDiscoveryArtefact` (i) reads the `DTDescriptor`s advertised by the WLDT platform on known endpoints, (ii) dynamically instantiates the corresponding CArtAgO artefacts by mapping entities in descriptors to CArtAgO artefacts' *observable properties* and *operations*, and (iii) sets up a persistent bi-directional connection to keep synchronised the artefact and the DT.

Thanks to this service, our `DTDiscoveryArtefact` can automatically create the artefacts corresponding to the available DTs by simply retrieving their descriptors and mapping elements therein to the appropriate CArtAgO abstraction: DT properties to artefacts *observable properties*, DT actions to artefacts *operations*, DT behaviours to both (each behaviour is an operation with additional lifecycle related observable properties), and DT relationships to *links* with other artefacts [18]. Metadata regarding the interaction protocol(s) are used to dynamically create a dedicated bi-directional communication "channel" between the artefact and the corresponding DT (e.g. a websocket or a sequence of REST request-response calls), so that the `DTDiscoveryArtefact` won't be a bottleneck by having to handle (collect and forward) all the interactions between all the DTs and all the artefacts. Such a link is meant to improve the scaling capabilities by decentralising the execution of communication action, without imposing a *tight coupling* between components. First, such a coupling only happens at runtime and fully automatically, without requiring design-time knowledge. Second, whenever such a bi-directional link fails (e.g. due to disconnections, components crashing, etc.), both the artefact and the DT may fall back to the mediation of the `DTDiscoveryArtefact` for the time needed to recover (e.g. restoring the communication link, replacing the faulty DT, etc.).

Each artefact dynamically created by the `DTDiscoveryArtefact` is added to the ad-hoc JaCaMo workspace `DTWorkspace` so that any agent in the MAS can discover and exploit them. The focal point of the whole integration just described is the `DTDiscoveryArtefact`, as it is the component that synergistically exploits existing JaCaMo and WLDT services (e.g. dynamic artefacts creation, in-workspace discovery, DT descriptors and their publication) to make totally transparent to the MAS developers the existence and utilisation of DTs. In fact, MAS developers need only to (i) configure the `DTDiscoveryArtefact` we designed with the well known address of `WLDTDiscoveryService`, and (ii) start the automatic "creation & mapping" process described above, by launching the dedicated operation provided by this library artefact. If the DTs layer is already up & running, the MAS environment will be automatically shaped accordingly. Moreover, as the `DTDiscoveryArtefact` is subscribed to changes in the `WLDTDiscoveryService`, newly created DTs will be promptly discovered and mapped to JaCaMo at run-time. Even agents with no prior knowledge (e.g. because some DTs are later added to the CPS) can discover what the environment has to offer by exploiting Jason reasoning capabilities and CArtAgO inspection services (e.g. get a list of available artefacts, get observable properties of an artefact, get its operations, etc.).

This openness and dynamism gives benefits in terms of separation of concerns and independent evolution, as MAS developers and DTs engineers can deal with their own part of the system using their preferred abstractions: MAS designers can think at the application as agents cooperating towards a given goal while interacting with available artefacts – regardless of how artefacts interact with physical entities –, whereas CPS engineers model physical resources and devices as DTs, and make their services (e.g. observing properties and requesting operations) available in a standard way (e.g. with web ready protocols and data formats).

## 4 Conclusion

In this paper, we outlined an integration architecture between MASs and DTs, to improve the way *environment engineering* is carried out in agent-oriented development practice, by exploiting the notion of DTs and their natural coupling with physical entities. In particular, we described such integration from both the conceptual and technical design perspective, relying on the A&A meta-model for the former, and on JaCaMo and WLDT development platforms for the latter.

With our proposal, greater separation of concerns both at run-time (between software components) and during design (between developers) is enabled, and system engineers can develop their own part of the system, the MAS and the CPS, independently. Implementation of the proposed design is already ongoing, as both JaCaMo and WLDT already offer most of the needed mechanisms. The `DTDiscoveryArtefact` will be release as a sort of "library artefact" ready to be used in any JaCaMo deployment.

## References


1. Boissier, O., Bordini, R.H., Hübner, J.F., Ricci, A., Santi, A.: Multi-agent oriented programming with JaCaMo. Science of Computer Programming **78**(6), 747–761 (2013)
2. Ciortea, A., Boissier, O., Ricci, A.: Engineering world-wide multi-agent systems with hypermedia. In: Weyns, D., Mascardi, V., Ricci, A. (eds.) Engineering Multi-Agent Systems - 6th International Workshop, EMAS 2018, Stockholm, Sweden, July 14-15, 2018, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11375, pp. 285–301. Springer (2018). https://doi.org/10.1007/978-3-030-25693-7_15
3. Croatti, A., Gabellini, M., Montagna, S., Ricci, A.: On the integration of agents and Digital Twins in healthcare. J. Medical Syst. **44**(9), 161 (2020). https://doi.org/10.1007/s10916-020-01623-5
4. Cruz Salazar, L.A., Ryashentseva, D., Lüder, A., Vogel-Heuser, B.: Cyber-physical production systems architecture based on multi-agent's design pattern—comparison of selected approaches mapping four agent patterns. The International Journal of Advanced Manufacturing Technology **105**(9), 4005–4034 (2019). https://doi.org/10.1007/s00170-019-03800-4
5. Gorodetsky, V., Kozhevnikov, S.S., Novichkov, D., Skobelev, P.O.: The framework for designing autonomous cyber-physical multi-agent systems for adaptive resource management. In: Marík, V., Kadera, P., Rzevski, G., Zoitl, A., Anderst-Kotsis, G., Tjoa, A.M., Khalil, I. (eds.) Industrial Applications of Holonic and Multi-Agent Systems - 9th International Conference, HoloMAS 2019, Linz, Austria, August 26-29, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11710, pp. 52–64. Springer (2019). https://doi.org/10.1007/978-3-030-27878-6_5
6. Iñigo-Blasco, P., Díaz-del-Río, F., Romero-Ternero, M.C., Cagigas-Muñiz, D., Diaz, S.V.: Robotics software frameworks for multi-agent robotic systems development. Robotics Auton. Syst. **60**(6), 803–821 (2012). https://doi.org/10.1016/j.robot.2012.02.004
7. Jennings, N.R.: An agent-based approach for building complex software systems. Communications of the ACM **44**(4), 35–41 (2001). https://doi.org/10.1145/367211.367250
8. Laird, J.E.: The SOAR cognitive architecture (2012)
9. Latsou, C., Farsi, M., Erkoyuncu, J.A.: Digital twin-enabled automated anomaly detection and bottleneck identification in complex manufacturing systems using a multi-agent approach. Journal of Manufacturing Systems **67**, 242–264 (2023). https://doi.org/10.1016/j.jmsy.2023.02.008, https://www.sciencedirect.com/science/article/pii/S0278612523000328
10. Latsou, C., Farsi, M., Erkoyuncu, J.A., Morris, G.: Digital twin integration in multi-agent cyber physical manufacturing systems. vol. 54, pp. 811 – 816 (2021). https://doi.org/10.1016/j.ifacol.2021.08.096
11. Liu, Z., Mao, X., Yang, S.: *AutoRobot*: A multi-agent software framework for autonomous robots. IEICE Transactions on Information and Systems **101-D**(7), 1880–1893 (2018). https://doi.org/10.1587/transinf.2017EDP7382
12. Mariani, S., Picone, M., Ricci, A.: About Digital Twins, agents, and multiagent systems: A cross-fertilisation journey. In: Melo, F.S., Fang, F. (eds.) Autonomous Agents and Multiagent Systems. Best and Visionary Papers - AAMAS 2022 Workshops, Virtual Event, May 9-13, 2022, Revised Selected Papers. Lecture Notes in Computer Science, vol. 13441, pp. 114–129. Springer (2022). https://doi.org/10.1007/978-3-031-20179-0_8

13. Martin, J., Casquero, O., Fortes, B., Marcos, M.: A generic multi-layer architecture based on ROS-JADE integration for autonomous transport vehicles. Sensors **19**(1), 69 (2019). https://doi.org/10.3390/s19010069
14. Minerva, R., Crespi, N.: Digital Twins: Properties, software frameworks, and application scenarios. IT Professional **23**(1), 51–55 (2021). https://doi.org/10.1109/MITP.2020.2982896
15. North, M.J., Collier, N.T., Ozik, J., Tatara, E.R., Macal, C.M., Bragen, M.J., Sydelko, P.: Complex adaptive systems modeling with Repast Simphony. Complex Adapt. Syst. Model. **1**, 3 (2013). https://doi.org/10.1186/2194-3206-1-3
16. Ocker, F., Urban, C., Vogel-Heuser, B., Diedrich, C.: Leveraging the asset administration shell for agent-based production systems. vol. 54, pp. 837 – 844 (2021). https://doi.org/10.1016/j.ifacol.2021.08.186
17. Omicini, A., Ricci, A., Viroli, M.: Artifacts in the A&A meta-model for multi-agent systems. Auton. Agents Multi Agent Syst. **17**(3), 432–456 (2008). https://doi.org/10.1007/s10458-008-9053-x
18. Omicini, A., Ricci, A., Zaghini, N.: Distributed workflow upon linkable coordination artifacts. In: Ciancarini, P., Wiklicky, H. (eds.) Coordination Models and Languages, 8th International Conference, COORDINATION 2006, Bologna, Italy, June 14-16, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4038, pp. 228–246. Springer (2006). https://doi.org/10.1007/11767954_15, https://doi.org/10.1007/11767954_15
19. Pedersen, S., Foss, B., Schjølberg, I., Tjønnås, J.: MAS for manufacturing control: a layered case study. In: van der Hoek, W., Padgham, L., Conitzer, V., Winikoff, M. (eds.) International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes). pp. 1169–1170. IFAAMAS (2012), http://dl.acm.org/citation.cfm?id=2343903
20. Picone, M., Mamei, M., Zambonelli, F.: WLDT: A general purpose library to build IoT digital twins. SoftwareX **13**, 100661 (2021). https://doi.org/10.1016/j.softx.2021.100661
21. Rao, A.S., Georgeff, M.P.: BDI agents: From theory to practice. In: Lesser, V.R., Gasser, L. (eds.) Proceedings of the First International Conference on Multiagent Systems, June 12-14, 1995, San Francisco, California, USA. pp. 312–319. The MIT Press (1995)
22. Ricci, A., Croatti, A., Mariani, S., Montagna, S., Picone, M.: Web of Digital Twins. ACM Transactions on Internet Technology **22**(4) (nov 2022). https://doi.org/10.1145/3507909
23. Ricci, A., Omicini, A., Denti, E.: Activity theory as a framework for MAS coordination. In: Petta, P., Tolksdorf, R., Zambonelli, F. (eds.) Engineering Societies in the Agents World III, Third International Workshop, ESAW 2002, Madrid, Spain, September 16-17, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2577, pp. 96–110. Springer (2002). https://doi.org/10.1007/3-540-39173-8_8, https://doi.org/10.1007/3-540-39173-8_8
24. Ricci, A., Piunti, M., Viroli, M., Omicini, A.: Environment programming in CArtAgO. In: Multi-Agent Programming, Languages, Tools and Applications, pp. 259–288. Springer (2009)
25. Russell, S.J., Norvig, P.: Artificial intelligence - a modern approach: the intelligent agent book. Prentice Hall series in artificial intelligence, Prentice Hall (1995), https://www.worldcat.org/oclc/31288015
26. Sakurada, L., Leitão, P., de la Prieta, F.: Engineering a multi-agent systems approach for realizing collaborative asset administration shells.

In: IEEE International Conference on Industrial Technology, ICIT 2022, Shanghai, China, August 22-25, 2022. pp. 1–6. IEEE (2022). https://doi.org/10.1109/ICIT48603.2022.10002770
27. Tao, F., Zhang, H., Liu, A., Nee, A.Y.C.: Digital Twin in industry: State-of-the-art. IEEE Trans. Ind. Informatics **15**(4), 2405–2415 (2019). https://doi.org/10.1109/TII.2018.2873186
28. Zekri, S., Jabeur, N., Gharrad, H.: Smart water management using intelligent Digital Twins. Comput. Informatics **41**(1), 135–153 (2022). https://doi.org/10.31577/cai_2022_1_135