Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnline CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 55-10 (2022) 1852–1857

Service Based Approach to Asset Administration Shell for Controlling Testing Processes in Manufacturing ⋆

Hamood Ur Rehman * **Jack C. Chaplin** * **Leszek Zarzycki** **
Fan Mo * **Mark Jones** ** **Svetan Ratchev** *

* *Institute for Advanced Manufacturing, University of Nottingham, Nottingham NG8 1BB, UK (e-mail: {Hamood.Rehman, Jack.Chaplin, Fan.Mo, Svetan.Ratchev}@nottingham.ac.uk)*
** *TQC Ltd., Hooton St, Carlton Rd, Nottingham NG3 2NJ, UK (e-mail: {leszek.zarzycki, mark.jones}@tqc.co.uk)*

Abstract: The rise in demand for customised products and lower costs has generated the need for incorporating intelligence in production systems. Intelligence is the main driver for enabling intelligent control in production systems to meet feature, part and conditional variations. It also promotes solutions that offer higher functionalities and lower costs. The research presented in this paper focuses on integrating intelligence into testing processes within a production system. A novel method of controlling the testing functionality is presented by deploying and using the asset administration shell approach. A means of controlling the process with the asset administration shell guided by simulation is demonstrated. An architecture for controlling the execution behaviour in testing process by services is conceptualised offering a modular and low-cost solution. A validation by using an experimental use-case of leak-testing is presented.

Keywords: leak testing, asset administration, low-cost industrial digitalisation and control

1. INTRODUCTION

Advances in computer engineering, communication, and information technologies are making a significant impact on manufacturing. "Intelligent manufacturing" is the term used to describe the next generation of production where production systems make intelligent decisions about themselves. It is expected that the use of intelligent manufacturing will expand significantly in the years ahead (Kostal et al. (2019)).

Intelligent manufacturing requires the application of technologies such as machine learning (Sharp et al. (2018)), reinforcement learning (Aggour et al. (2019)) and cloud computing (Rehman et al. (2021c)) to enable devices and machines to adjust their behaviour in response to diverse circumstances and requirements, based on previous experiences and learning capabilities (McFarlane et al. (2003)). These technologies can be integrated to make intelligent decisions in production processes (Zhong et al. (2017)). Some reconfiguration frameworks that incorporate this integration are proposed to meet product change requirements (Rehman et al. (2021b)).

The Reference Architectural Model Industrie 4.0 (RAMI 4.0) (Schweichhart (2016)), a reference model for Industry 4.0 (I4.0), recommends that manufacturing "assets" can be represented by digital objects called "*Asset Administration Shell (AAS)*", that enables standardised interaction with other digital systems without requiring low-level access to sensors and actuators. The goal of I4.0 AAS is to offer a minimum but sufficient description of each Industry 4.0 asset, that can then be manipulated and influenced to achieve an objective.

This concept of AAS is at its initial stage in the I4.0 paradigm and needs validation through industrial use-cases to demonstrate that it can be successful in production process control. This validation is also vital to elaborate on the needs of production process control that must be fulfilled by an AAS. To date, three types of AAS can be defined based on the active/proactive behaviour of the desired solution:

- Type 1 Asset Administration Shells are serialised files, e.g., XML, or JSON files, that contain static information.
- Type 2 Asset Administration Shells exist as runtime instances. They are hosted on servers, contains static information, and may also interact with other components.
- Type 3 AAS extend type 2 AAS by implementing an active behaviour, i.e., they can start to communicate and to negotiate autonomously, much like an agent system.

⋆ This work is carried out under DiManD Innovative Training Network (ITN) project funded by the European Union through the Marie Sktodowska-Curie Innovative Training Networks (H2020-MSCA-ITN-2018) under grant agreement number no. 814078.

10.1016/j.ifacol.2022.09.668

This research work presents a demonstration case for the potential of Type 2 AAS on production systems, guidance on integration with I4.0 technologies and the objectives to be fulfilled in an AAS to achieve control. It addresses the practical aspect of AAS by demonstrating the implementation of AAS on a testing process in a production system. A testing process is a manufacturing process where a part or product is subjected to conditions it might encounter during service, to ensure correct functionality. Examples include functional testing of electrical circuits, stress testing of assemblies, and gas or liquid leak testing of volumes. Testing processes are particularly important for small to medium enterprises (SMEs), who often provide low-volume, highly bespoke products. Testing processes are highly dependent on the part or product being tested, and this can become difficult for companies to manage for small batches of variable products. This motivates a call to digitalise and control these processes with low-cost, simple, and standards-compliant solutions.

In this research work, it is assumed that a serialised file of Type 1 AAS is present (in JSON or XML) to represent the asset and can be instantiated as run-time instance (registered in AAS registry, hosted on server, and can load data). This research work focuses on behaviour change on the asset, by utilising services that use the AAS to direct control behaviour of a production process.

2. BACKGROUND

As depicted in Figure 1, an administrative shell is composed of four major components (Wenger et al. (2018)), namely: component manager, manifest, header section and body section.

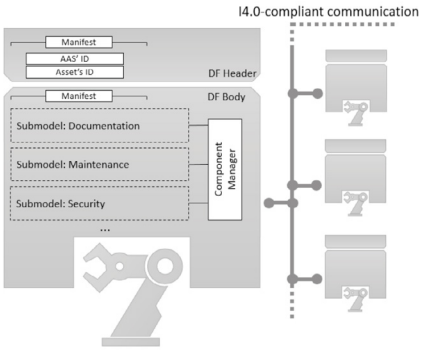

Fig. 1. Asset Administration Shell representation of an asset with its main components for I4.0 compliant communication with IoT infrastructure within a production facility.

- **The DF (digital factory) Header Section** contains the globally unique identifiers for an AAS and its represented asset.
- **The DF Body Section** is composed of multiple submodels, each representing a distinct part of the asset's operation.
- **Component Manager** links the administration shell to a repository of submodels, their description and their functions. It administers the submodels of the assets. The Component Manager manages and provides access to the IoT network of the production facility using a service-oriented architecture.
- **Manifest** is present in both header and body section of AAS. It can be considered as the directory of data content. Specifically, it contains the meta-information serving to provide meaning to the data from AAS.

Submodels depict the various facets of an asset, represented as per the standard discussed in Bedenbender et al. (2017). These submodels contain the information encapsulating the asset's functionality, i.e., an asset can have submodels for maintenance, operation, and security among others. In a production facility, the AAS can be connected to a network that acts as a bridge between the physical world and the digital. Compatible AAS protocols and channels can be queried through services, and these services can then use the AAS to execute behaviour (Wagner et al. (2017); Cavalieri and Salafia (2020b)).

Although the structure of AAS is becoming standardised, the method by which they are used remains variable. In Cavalieri and Salafia (2020a), a standard method to realise AAS is studied, formulating an AAS model capable of representing IEC 61131-3 programmes, their interactions with Programmable Logic Controllers (PLCs), and other components in the controlled plant. In Tantik and Anderl (2017) a framework for connecting existing equipment to external networks is introduced; the asset management shell serves as a bridge simplifying network communication. In Pethig et al. (2017), an AAS approach is proposed that integrates with a condition monitoring service. In Wenger et al. (2018), when considering the possible impact on PLC operation, a distributed AAS is utilised to enhance performance. There are several ways to use the AAS, as discussed in the work by Löcklin et al. (2021).

AAS is a key enabler of the digitalisation of processes, and do so in a low-cost, modular, and standards-compliant way that can be adopted by SMEs. Currently, AAS has some limitations, particularly a lack of a standardised approach for direct adoption of AAS into production processes (Tantik and Anderl (2017)). A lack of examples of their application in controlling production processes in production systems also hinders adoption. This research work addresses this lack of applications and standardisation in production processes by targeting testing processes in a production system in manufacturing.

3. SERVICE BASED CONTROL OF ASSET ADMINISTRATION SHELL

To intelligently control testing processes, a service-based control framework is used that integrates asset administration shells. This framework would support the standardised application of AAS to manufacturing systems. This framework requires interaction between five components; the testing system, asset administration shell, simulation service, client services and orchestration service. The *orchestration service* drives and controls the asset by a statement received from the simulation service acting on a goal provided by the user. The *simulation service* result is sent to the orchestration service at the start of each test, and the orchestration service executes the testing process through *client services*.The client services interact with *AAS* to execute *testing* functionality. The client services receive the instructions from the orchestration service, loads required information from the AAS, makes changes

to the production system as desired and executes the skills through API (Application Programming Interface) calls retrieved from the AAS functionality submodel. These calls act as instructions to the asset to execute a skill (corresponding to a step of the testing operation). An overview of the asset administration components and their interaction is given in the following subsections.

3.1 Component Descriptions of the Asset Administration Shell and Services in the Approach for testing Process

A brief overview dealing AAS integration with the testing process for skill execution is presented in figure 2. The services drive the functionality of the production system by coordinating among themselves and by interacting with information from the asset administration shell. Individual components and their functionalities of the services and the AAS are detailed as follows;

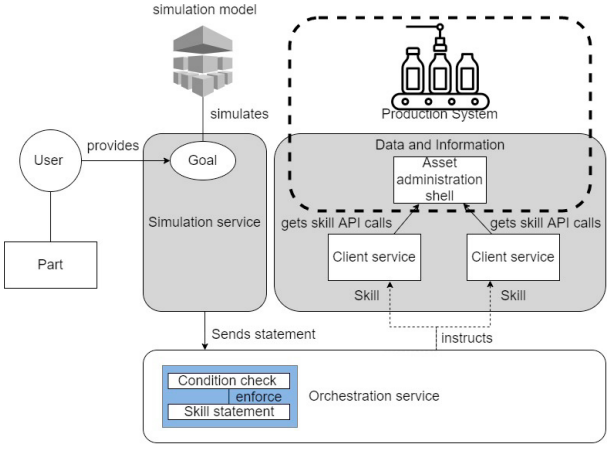

Fig. 2. Elaborated representation of asset administration shell integration for testing process in production system.

Testing Process As defined in the introduction, the testing process subjects a part to operational conditions to check functionality. Part, feature, and conditional variation (i.e., changing priorities and requirements) can impact the operation of testing processes. Further, tests need to be carried out in a certain sequence with specific conditions. The sequence includes steps such as adjusting the configuration and executing the test, but may also include aspects such as establishing connections with other assets and accessing data. Conditions act as parameters for each of the steps in the sequence. This may include how to set and change testing parameters (e.g., within a certain range), the number of iterations for the testing, or the address of data sets for comparison.

Asset Administration Shell The server shell contains all the information about the physical asset. It includes the related parameters, expressions, configurations, and settings that represent the physical entity. It also contains the API calls necessary for executing the skills. It is the digital representation of the asset.

As the AAS is instantiated, it is registered in the registry (if present for deployment) and hosts its submodels managed by the Component Manager (figure 3 and 6).

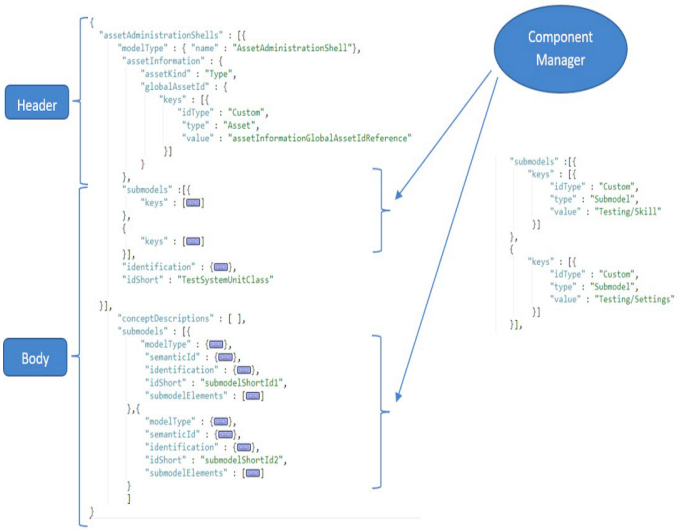

Fig. 3. Generic asset administration shell representation *(left)*. Skill submodel is developed, submodel elements contains the skill API calls along with pertaining data that can be used by the client services. Component manager is deployed when the submodels are listed *(right)*.

Simulation Service The simulation service connects to the simulation model. External or local servers can host the simulation model, connected to a storage or a Machine Learning pipeline. The model consists of a clearly, defined goal as well as start and intermediary states. The model simulates the path to reach the goal in the optimum manner. The simulation service takes the decision to move between intermediary states to reach goal states (figure 4).

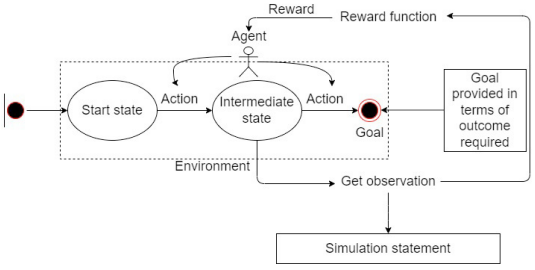

Fig. 4. Generating the *simulation statement* by simulation service. The model operates to reach the goal provided dependent on the outcome required. The intermediate states are stored and combined to formulate a simulation statement that is sent to the orchestration service.

A reward function uses performance metrics to determine if the action taken moves the state towards the goal. The metrics used could include time, cost, energy consumption, and quality.

The actions defined within the simulation service act on the model to reach the goal guided by the reward function. Simulation continues until an end point is reached, namely the provided conditions or goal become true. The efficacy of the simulation depends on the design of the reward function.

Orchestration Service The orchestration service guides the execution of skills on the asset through client services (figure 5). It receives the simulation statement and transforms it to an orchestration statement. The statements are expressions that contain the information the services need for execution of a skill. The process of developing these, the governing rules, and their behaviour are explained in Rehman et al. (2021a). The Orchestration service passes the orchestration statement through its two main components, the condition check and the skill execution.

(1) **Condition Check:** For each individual skill requested, the service establishes the availability of that particular skill for the testing process. This is queried from the asset administration shell by the orchestration service. A sequence of executable skills are sent to the next component of the orchestration service.
(2) **Skill Execution:** The skills are executed in the presented sequence. This execution is carried out by the client service for each skill. The skill execution component finds and instructs the respective client service to execute the skill for the testing process.

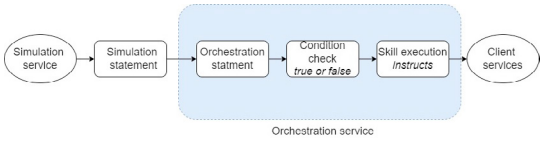

Fig. 5. Orchestration service description with its components.

Client Services The client services encapsulate the skill execution and the manner of skill execution. The client services requests from the AAS the API calls needed to execute the skills. These are contained within the corresponding property of skill submodel along with the pertaining data values from the submodel settings (figure 3). Client services execute these skill API calls. They can also change the settings on an asset and update them as needed in the AAS and the production system, ensuring real-time representation of the settings of the production system in the AAS. Client services offer a way to control the behaviour of skill execution. These are called "client services" as they act as a liaison between the execution functionality of the testing process and the asset administration shell. Client services are separate software components that interact with the AAS to execute test functionality as per guidance from the orchestration service. The stand-alone nature of client services ensure modularity and versatility in terms of services being offered.

3.2 Overview of Skill Execution in testing Processes

The execution of skills in a testing process by using AAS and services can be presented as follows;

Skill Representation Each of the skills (corresponding to a step of the testing operation) is encapsulated as a property of the "Skill" submodel of the AAS. These skills are operations that an asset can be requested to perform in the testing process. The properties of the AAS in the submodel represent the skill of the asset that could be executed in the form of API calls. It also contains the information about the data on skill that could be executed, present in the manifest.

Skill Selection The skill order is controlled by the orchestration service through a statement received by the simulation service. The client services get the API calls through the property of the skill submodel and the related data values and use them to execute behaviour requested by the orchestration service. Each client service is responsible for one skill per asset. These client services also contain information about behaviour of the particular skill. An overview of the AAS integration architecture is presented in figure 6.

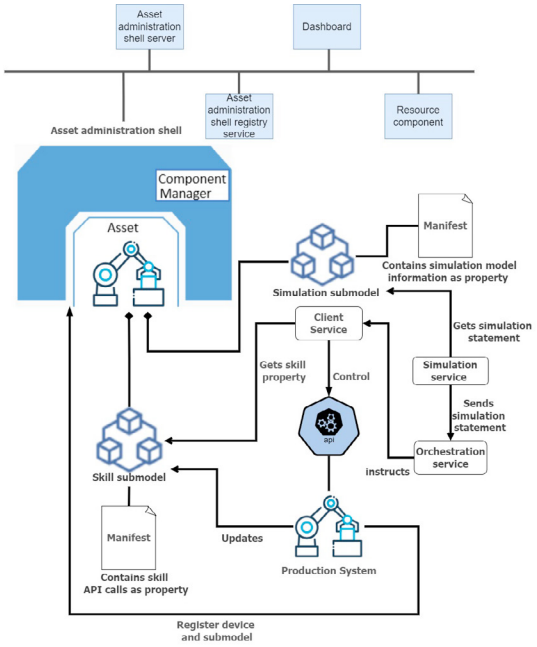

Fig. 6. Asset administration shell integration approach for testing processes. Control is guided through services, through Skill and Simulation submodels containing API calls and simulation data respectively

Skill Execution The client services execute skills through API calls retrieved from the skill submodel. System functionality is achieved through this execution. The AAS contain a set of skills that are available to execute. Client services load instances of these skills as required. The client service, instructed by the orchestration service, requests the execution of skill using the API on the production system and waits until the execution is completed. After the event of skill execution, the production system updates the status of skill execution in the property of the skill submodel. This terminates the client skill request and orchestration service proceeds to the next skill execution through a corresponding client service. This integration of AAS submodels and client services ensures modularity, scalability and extensibility. For a single asset there is a single AAS but many client services where each client service represents a skill that could be triggered by loading the API calls from the AAS. Each client service, however, can trigger only one skill.

The orchestration service, operating as a standalone service, executes the skills through client services, updating

Table 1. Asset administration shell and services description for testing process in production system

Component	Description
Simulation Service	Simulates the desired goal behaviour. Send simulation statement.
Orchestration Service	Receives orchestration statement. Checks and executes the skill functionality through client services.
Client Services	Receives instruction to execute from the orchestration service and loads in the required settings in the desired manner. Executes the skill API calls from the asset administration shell.
Asset Administration Shell	Contains the data and information about the asset. Hosts the skill API calls. This shell acts as a digital twin for the asset.

regularly to account for different requirements. These requirements can be a change in sequence of skill execution or the manner of skill execution, such as skill execution with a value change. This update can be done manually by the user or received as a result from simulation service.

Scenarios such as part variation, feature, and conditional variation can be addressed by this approach. Figure 7 illustrates the AAS and the services in each scenario and the manner in which they co-ordinate for skill execution during operation.

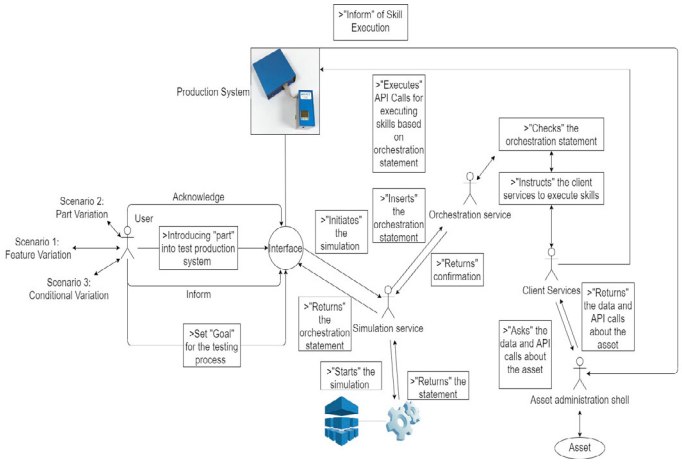

Fig. 7. Role of AAS integration approach in functionality execution with changing scenarios; part variation, feature variation and conditional variation

The AAS and services interact at different stages of the testing process, depending on the requirements that need to be fulfilled. These requirements, as fulfilled, can be witnessed as the result of the testing or as the change in the values in settings.

4. INTEGRATION APPROACH DEPLOYMENT FOR A LEAK TESTING USE CASE

The demonstration for the asset administration shell integration is carried out on an industrial dry-air leak testing system (Micro-Application Leak Test System – MALT) developed by an SME (TQC Ltd.). The industrial production system consists of a testing system connected to a Raspberry Pi that acts as a gateway device. A JAVA based AAS development and deployment framework (BaSyx (BaSyx (2022))) is used for to demonstrate the approach.

In the process of leak testing, the leakage flow rate in the part under test is determined. Parameters for leak-testing, connection settings for the production system and calibration settings for the connected instrumentation are configured through a user interface. The testing process is initialised with filling the part under observation with testing medium or put in a state of vacuum. The part is stabilised over time under pressure or vacuum. The change in pressure is measured and is used to calculate the leakage flow rate in the part being tested. Pressure change or its proportional leakage flow rate value is used to determine pass or fail for the part.

The goal of the test is provided by the user to the simulation service that generates the simulation statement. This statement is fed to the orchestration service. An example of simulation statement from the simulation service is:

$$\begin{aligned} Simulation\ statement =& < basic : 192.168.115.200 : \\ & 5000 > . < condition : MaxTestPressure\ 15500 > . \\ & < stab : 2000 > . < fill : 3000 > . \\ & < param : measure : 8000 > . < execute > \end{aligned} \quad (1)$$

In leak testing, we provide the goal to "execute" a test. Conditions were provided to the simulation. The manner or sequence of execution are followed by the combination passed from the simulation statement. The orchestration statement acts as a soft controller that guides it to execute the skills, moving from one state to another.

$$\begin{aligned} Orchestration\ Statement =& [basic].[condition]. \\ & [stab].[fill].[param].[execute] \end{aligned} \quad (2)$$

The "basic" client service starts the connection with the testing system, the "condition" service uses conditions to change the testing parameters, and stabilisation time is adjusted through "stab" client service. The "fill" and "param" client service change the fill time and any parameters (taken as argument) respectively. The "execute" client service executes the test on the target part. Giving a client service responsibility for individual skills gives the capability to define testing for different parts depending on the statement; presented by the simulation service.

The client services contain the information on the manner in which each skill for the leak testing needs to be executed. The client service take in leak testing data from AAS submodel and produce change as desired. The desired change in settings by each client service is loaded through an API call from the AAS skill submodel, execution of the leak testing skill is instantiated. The leak testing process is carried out until the orchestration service reaches the complete state. The execution interface for testing process is given in figure 8.

This approach provides a control mechanism for executing testing processes controlled through services. The services utilise the Type 2 AAS representation to produce behavioural change, directing control of the testing process. The AAS deployed for leak testing follows the AAS standard, making it extensible and modular. As the approach uses a single AAS, so it makes the approach modular and cost-effective as multiple individual component requirements need not be addressed.

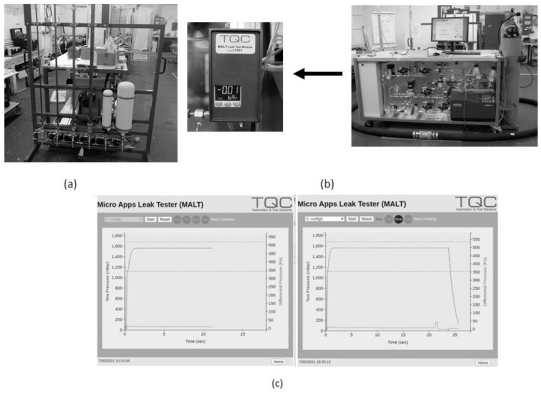

Fig. 8. The leak testing setup: (a) the cylinder volumes under testing (b) MALT testing system, a part of test bench for general leak testing (c) Interface for leak testing; asset administration shell drives the execution through client services.

5. CONCLUSION AND FUTURE WORK

Testing processes are highly variable depending on the part or product being tested. Successful testing is typically dependent on human experts and manual changes due to the complex process sequences and conditions. This makes testing processes prime candidates for digitalisation and automation to reduce set-up time, costs, and errors. Many SMEs are highly dependent on testing processes to validate their low batches of high-value and bespoke parts or products. Any digital solutions must therefore be low-cost to maximise impact, especially for SMEs. AAS is the coming standard for digitalising manufacturing assets and maximising modularity and interoperability. A static AAS is inadequate for adapting to changing testing needs, the type 2 is required.

This research presents an application case for integration approach towards Type 2 AAS with testing process. The research contributes towards controlling the functionality of a testing process in production systems by using services and AAS. The presented approach significantly reduces the time required to set up a new testing process and requires less expert time to control the testing operation.

Three different types of services were presented in this research: simulation service, orchestration service and client services. The simulation service takes in a goal from the user and acts on a model to define the best possible decision for the orchestration service in terms of skills that need to be executed to reach the goal state. The orchestration service is responsible for executing the skills by instantiating the client services. The client service interacts with the AAS to execute the skills. The development and deployment approach for AAS is presented, along with integration with services.

Future work for this approach involves expanding the submodel architecture to incorporate other service components. Reinforcement Learning and Machine Learning techniques can be explored for achieving multiple goals as desired. Service negotiation behaviour and transition approach from Type 2 AAS to Type 3 AAS will be explored.

REFERENCES

Aggour, K.S., Gupta, V.K., Ruscitto, D., Ajdelsztajn, L., Bian, X., Brosnan, K.H., Kumar, N.C., Dheeradhada, V., Hanlon, T., Iyer, N., et al. (2019). Artificial intelligence/machine learning in manufacturing and inspection: A ge perspective. *MRS Bulletin*, 44(7), 545–558.

BaSyx, E. (2022). Eclipse basyx. *Enabling Open Innovation & Collaboration*.

Bedenbender, H., Billmann, M., Epple, U., Hadlich, T., Hankel, M., Heidel, R., Hillermeier, O., Hoffmeister, M., Huhle, H., Jochem, M., et al. (2017). Examples of the asset administration shell for industrie 4.0 components–basic part. *ZVEI white paper*.

Cavalieri, S. and Salafia, M.G. (2020a). Asset administration shell for plc representation based on iec 61131–3. *IEEE Access*, 8, 142606–142621.

Cavalieri, S. and Salafia, M.G. (2020b). Insights into mapping solutions based on opc ua information model applied to the industry 4.0 asset administration shell. *Computers*, 9(2), 28.

Kostal, P., Mudrikova, A., and Michal, D. (2019). Possibilities of intelligent flexible manufacturing systems. In *IOP Conference Series: Materials Science and Engineering*, volume 659, 012035. IOP Publishing.

Löcklin, A., Vietz, H., White, D., Ruppert, T., Jazdi, N., and Weyrich, M. (2021). Data administration shell for data-science-driven development. *Procedia CIRP*, 100, 115–120.

McFarlane, D., Sarma, S., Chirn, J.L., Wong, C., and Ashton, K. (2003). Auto id systems and intelligent manufacturing control. *Engineering Applications of Artificial Intelligence*, 16(4), 365–376.

Pethig, F., Niggemann, O., and Walter, A. (2017). Towards industrie 4.0 compliant configuration of condition monitoring services. In *2017 ieee 15th international conference on industrial informatics (indin)*, 271–276. IEEE.

Rehman, H.U., Chaplin, J.C., Zarzycki, L., Jones, M., and Ratchev, S. (2021a). Application of multi agent systems for leak testing. In *2021 9th International Conference on Systems and Control (ICSC)*, 560–565. IEEE.

Rehman, H.U., Chaplin, J.C., Zarzycki, L., and Ratchev, S. (2021b). A framework for self-configuration in manufacturing production systems. In *Doctoral Conference on Computing, Electrical and Industrial Systems*, 71–79. Springer.

Rehman, H.U., Pulikottil, T., Estrada-Jimenez, L.A., Mo, F., Chaplin, J.C., Barata, J., and Ratchev, S. (2021c). Cloud based decision making for multi-agent production systems. In *EPIA Conference on Artificial Intelligence*, 673–686. Springer.

Schweichhart, K. (2016). Reference architectural model industrie 4.0 (rami 4.0). *An Introduction. Available online: https://www. plattform-i40. de I*, 40.

Sharp, M., Ak, R., and Hedberg Jr, T. (2018). A survey of the advancing use and development of machine learning in smart manufacturing. *Journal of manufacturing systems*, 48, 170–179.

Tantik, E. and Anderl, R. (2017). Potentials of the asset administration shell of industrie 4.0 for service-oriented business models. *Procedia CIRP*, 64, 363–368.

Wagner, C., Grothoff, J., Epple, U., Drath, R., Malakuti, S., Grüner, S., Hoffmeister, M., and Zimermann, P. (2017). The role of the industry 4.0 asset administration shell and the digital twin during the life cycle of a plant. In *2017 22nd IEEE international conference on emerging technologies and factory automation (ETFA)*, 1–8. IEEE.

Wenger, M., Zoitl, A., and Müller, T. (2018). Connecting plcs with their asset administration shell for automatic device configuration. In *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, 74–79. IEEE.

Zhong, R.Y., Xu, X., Klotz, E., and Newman, S.T. (2017). Intelligent manufacturing in the context of industry 4.0: a review. *Engineering*, 3(5), 616–630.