Journal of Manufacturing Systems 72 (2024) 424–436

Contents lists available at ScienceDirect

Journal of Manufacturing Systems

journal homepage: www.elsevier.com/locate/jmansys

Technical paper

A modular artificial intelligence and asset administration shell approach to streamline testing processes in manufacturing services

Hamood Ur Rehman [a,b,1], Fan Mo [a,*,1], Jack C. Chaplin [a], Leszek Zarzycki [b], Mark Jones [b], Svetan Ratchev [a]

[a] Institute for Advanced Manufacturing, University of Nottingham, Nottingham, Nottinghamshire, NG8 1BB, United Kingdom
[b] TQC Automation Ltd., Nottingham, Nottinghamshire, NG3 2NJ, United Kingdom

ARTICLE INFO

Keywords:
Leak testing
Asset administration
Low-cost industrial digitalization and control
Modular artificial intelligence

ABSTRACT

The increasing demand for personalized products and cost-effectiveness has highlighted the necessity of integrating intelligence into production systems. This integration is crucial for enabling intelligent control that can adapt to variations in features, parts, and conditions, thereby enhancing functionalities while reducing costs. This research emphasizes the incorporation of intelligence in testing processes within production systems. We introduce a novel approach for controlling testing functionality using an asset administration shell enriched with modular artificial intelligence. The proposed architecture is not only effective in controlling the execution behavior through services but also offers the distinct advantage of a modular design. This modularity significantly contributes to the system's adaptability and scalability, allowing for more efficient and cost-effective solutions as different machine-learning models may be substituted to meet requirements. The effectiveness of this approach is validated through a practical use case of leak testing, demonstrating the benefits of the modular architecture in a real-world application.

1. Introduction

The fourth industrial revolution, also known as Industry 4.0 (I4.0), continuously transforms the manufacturing landscape. The convergence of computer engineering, communication, and information technologies fosters a revolution within manufacturing systems that propels the industry into a new era of intelligent manufacturing [1,2]. This modern approach integrates advanced technologies such as artificial intelligence, the Internet of Things, big data analytics, knowledge graph and cloud computing to create a highly efficient and productive manufacturing ecosystem [3,4].

Intelligent manufacturing requires the application of technologies such as machine learning [5], reinforcement learning [6], and cloud computing [7] to enable devices and machines to adjust their behavior in response to diverse circumstances and requirements, based on previous experiences and learning capabilities [8]. These technologies can be integrated to make intelligent decisions in production processes [9,10]. Some reconfiguration frameworks that incorporate this integration are proposed to meet product change requirements [11,12].

In the context of I4.0, the Reference Architectural Model Industrie 4.0 (RAMI 4.0), as per Schweichhart's Ref. [13], is a pivotal model. These objects are known as the Asset Administration Shell (AAS). The AAS enables standardized interaction with other digital systems. This interaction does not require low-level access to sensors and actuators. The goal of the AAS in I4.0 is to provide a minimal but sufficient description of each asset. These descriptions can then be manipulated and influenced to achieve a specific objective.

This concept of AAS is at its early stage of adoption 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 [14]:

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

* Corresponding author.
E-mail address: fan.mo@nottingham.ac.uk (F. Mo).
[1] These authors contributed equally to this work.

https://doi.org/10.1016/j.jmsy.2023.12.004
Received 20 August 2023; Received in revised form 23 November 2023; Accepted 13 December 2023
Available online 23 December 2023

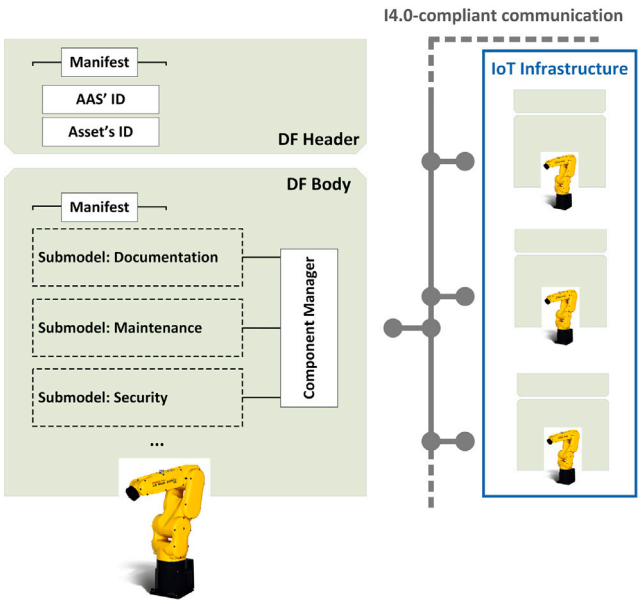

Fig. 1. Asset administration shell representation of an asset with its main components for I4.0 compliant communication with IoT infrastructure within a production facility.

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 AASs in 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) [10], which often provide low-volume, highly bespoke products. Testing processes are highly dependent on the part or product being tested, and it can become difficult for companies to manage small batches of variable products. This motivates a call to digitalize and control these processes with low-cost, simple, and standards-compliant solutions.

For this research, it is assumed that a serialized file of Type 1 AAS is present (in JSON or XML) to represent the asset and can be instantiated as a run-time instance (registered in the AAS registry, hosted on the server, and can load data). This research project works to examine changes in behavior regarding assets by utilizing services (Machine Learning) that use the AAS to control the behavior of a production process. This research builds upon our previously published conference paper titled "Service-based Approach to Asset Administration Shell for Controlling Testing Processes in Manufacturing", which was presented at the 10th IFAC Conference on Manufacturing Modelling, Management, and Control (MIM 2022) [15].

The structure of the paper is as follows: Section 1 introduces the concept of asset administration shell and presents the research work, Section 2 presents some background work, Section 3 elaborates on the concept of integration of asset administration shells with control services, and Section 4 demonstrates the application on a leak testing use-case. Finally, Section 5 concludes the work and outlines future directions.

2. Background

2.1. Asset administration shell

As depicted in Fig. 1, an administrative shell is composed of four major components [16], namely: component manager, manifest, header section, and body section.

- **The Digital Factory (DF) 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.
- **The Component Manager** links the AAS 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 Internet of Things (IoT) network of the production facility using a service-oriented architecture.
- **The Manifest** is present in both the header and body sections 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 [17]. 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 behavior [18,19].

Although the structure of AAS is becoming standardized, the method by which they are used remains variable. In [20], a standard method to realize AAS is studied, formulating an AAS model capable of representing IEC 61131-3 programs, their interactions with Programmable Logic Controllers (PLCs), and other components in the controlled plant. In [21], a framework for connecting existing equipment to external networks is introduced; the asset management shell serves as a bridge simplifying network communication. In [22], an AAS approach is proposed that integrates with a condition monitoring service. In [16], when considering the possible impact on PLC operation, a distributed AAS is utilized to enhance performance. There are several ways to use the AAS, as discussed in the work by [23].

AAS is a key enabler of the digitalization of processes and does 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 standardized approach for the direct adoption of AAS into production processes [21]. 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 standardization in production processes by targeting testing processes in a production system in manufacturing.

2.2. Modular AI and tabular machine learning

Modular AI is an approach in the realm of artificial intelligence that breaks down a complex AI system into smaller, specialized components, often referred to as modules [24]. Each of these modules is designed to handle a specific task or a set of related tasks. This can range from various forms of data processing to more intricate functionalities such as decision-making or prediction. The modular structure facilitates a manageable design, efficient implementation, and effective management of complex AI systems. One of the key advantages of this approach is its inherent flexibility and extensibility [25], which allows for the integration of a variety of techniques, methodologies, and technologies, including emerging machine learning methods.

Tabular Machine Learning (TML) is one such method that can be seamlessly integrated into a modular AI system. TML refers to the application of machine learning models to structured, tabular data. This form of data consists of rows, representing individual observations, and columns, indicating different features or attributes of these observations. This type of data is ubiquitous and can be found in various fields such as finance, healthcare, and manufacturing, among others [26].

In TML, machine learning models, including but not limited to, linear regression [27], logistic regression [28], decision trees [29], and gradient-boosting machines [30], are applied to predict a target variable based on a set of features. The choice of these models is dependent on the problem's specific requirements, such as the nature of the target variable, the data distribution, and the model's interpretability.

Furthermore, TML often involves a process known as feature engineering [31]. This process involves creating new, more informative features from the existing ones to improve the model's predictive performance. This often requires domain knowledge to develop meaningful features.

When integrated within a modular AI system, TML acts as a modular component that can handle specific tasks related to structured, tabular data. For instance, in the context of manufacturing, a TML module could be developed to predict the outcome of leak tests based on various parameters such as product dimensions, material properties, and manufacturing conditions.

The integration of TML within a modular AI system is not only about using TML to solve a specific problem but also about the flexibility it offers. By treating TML as a separate, interchangeable module, it is possible to switch between different datasets or even machine learning models, enhancing the adaptability and versatility of the AI system.

In this research, we delve deeper into the application of TML for leak testing, where it serves as a modular component within a larger AI system. In this application, an interchangeable dataset or a modular machine learning model can be employed to predict the leak testing outcomes, demonstrating the power of the Modular AI approach. This also offers opportunities for further research and development, as new techniques or algorithms can be incorporated into the system as separate modules, allowing the system to evolve and improve over time.

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 can support the standardized 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 *testing system* performs the functionality, while *AAS* contains all information on the asset. The *simulation service* generates a simulation statement by interacting with a model to determine the best possible settings for the functionality based on goals provided by the user. The *client services* execute the functionality by interacting with the production system and the AAS using the orchestration service. The *orchestration service* drives and controls the asset by guiding the execution of functionality through client services (see Table 1).

The client services receive the instructions from the orchestration service, load required information from the AAS, make changes to the production system as desired, and execute 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

The AAS integration with the testing process is illustrated briefly in Fig. 2. The services drive the functionality of the production system by coordinating among themselves and by interacting with information from the asset administration shell. There can be multiple client services interacting with the asset administration shell. E.g., a simulation service can be a client service that is responsible for simulation along with a functionality service responsible for executing the operation. If multiple operations are possible by the systems, like a robot capable of marking and gripping, each will be represented by a different client service. Individual components and their functionalities of the services and the AAS are detailed as follows;

3.1.1. Testing process

As defined in the introduction, the testing process subjects a part to operational conditions to test it for a pass/fail based on criteria. Part, feature, and conditional variations (i.e., changing priorities and requirements) are examples of operation conditions that impact the operation of the testing process. Further, these tests must be carried out in a certain sequence with specific conditions per product/customer requirements. 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.

3.1.2. Asset administration shell

The server shell contains all the information about the physical asset. It includes the related parameters, expressions, configurations, and settings representing 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 (Figs. 3 and 8).

3.1.3. Simulation service

The simulation component consists of a digital twin, acting as a digital representation of the physical asset in the production system. This is the data that goes from the physical asset to the simulation environment. Data is represented using AAS, capturing the production system information. The simulation environment supports multiple data representations for different assets and employs optimization algorithms to determine the best possible configurations under set key performance indicators (KPIs). The optimized configuration is then loaded into the production system for execution. Communication between the physical and digital representations and optimization library can be established using protocols like OPC UA, MQTT, or TCP/IP.

The KPI is set within the simulation environment, which then selects the appropriate ML model. Using the optimization library, the simulation environment determines the optimal values for specific

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

Component	Description
Simulation service	Uses modular ML AI models to simulate goal behavior and sends simulation statements.
Orchestration service	Receives orchestration statements and coordinates execution of skills through client services.
Client services	Executes skills using API calls obtained from the asset administration shell based on instructions from the orchestration service.
Asset administration shell	Acts as a digital twin for the asset, hosting skill API calls and storing asset data.

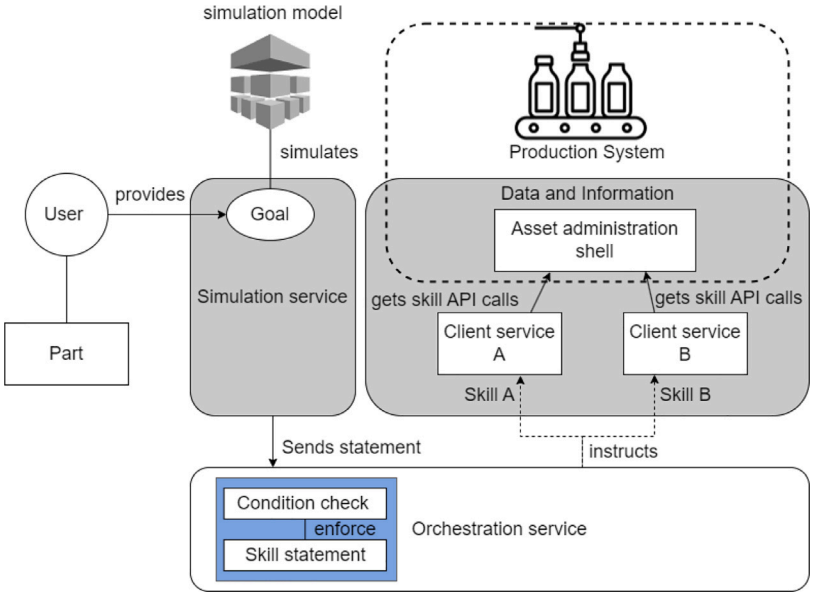

Fig. 2. Elaborated representation of asset administration shell integration for testing process in the production system.

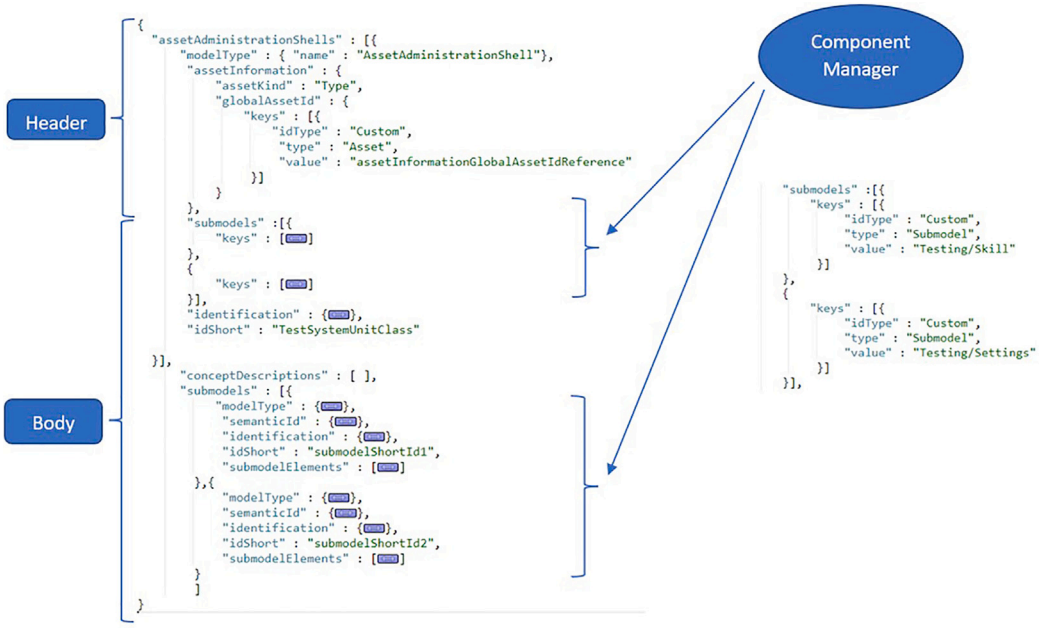

Fig. 3. Generic asset administration shell representation *(left)*. Skill submodel is developed, submodel elements contain the skill API calls along with pertaining data that can be used by the client services. The component manager is deployed when the submodels are listed *(right)*.

functionality. The resulting values are then received and loaded into the production system, enabling the execution of the desired functionality.

The simulation service hosts the digital twin as a simulation model. External or local servers can host the simulation model, connected to storage or a Machine Learning pipeline (see Fig. 4).

A reward function uses performance metrics to determine if the action taken moves the production system state towards the optimization 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 endpoint is reached. Namely, the provided conditions or goal become true. The efficacy of the simulation depends on the design of the reward function.

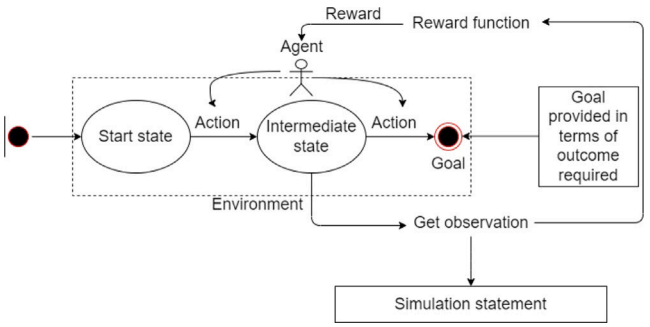

Fig. 4. Generating the *simulation statement* by simulation service. The model operates to reach the goal provided, dependent on the outcome required. The intermediate states are stored and combined to formulate a simulation statement sent to the orchestration service.

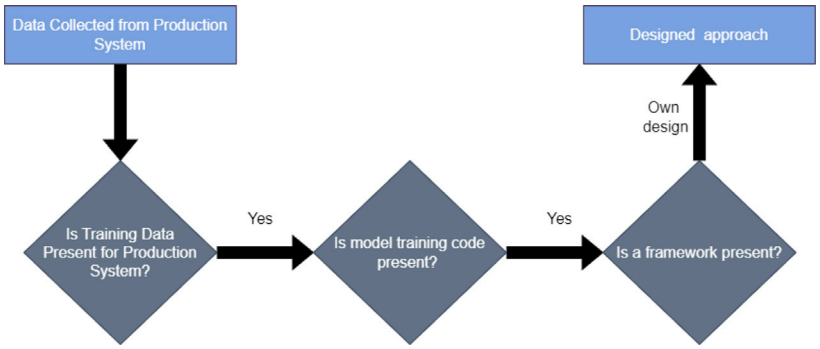

Fig. 5. Proposed workflow for machine learning in the production system for low-powered production systems.

3.1.4. Modular AI

The proposed architecture aims to provide machine learning capabilities for predicting the values of variables in parameters to execute functionality in testing systems. Workflows are introduced as a generalized approach to applying machine learning in production system use cases, focusing on efficient results for value determination. These workflows serve as a foundation for integrating machine learning services, providing an abstract viewpoint while keeping the implementation modular and emphasizing the overall implementation rather than ML component specifications. TML is one example of such workflow that is deployed in the research.

Accumulating data from the production system is crucial, and relevant inputs and outputs must be defined as features. The approach utilizes the availability of off-the-shelf training code to determine values. As the training code exists, standard off-the-shelf approaches or custom-designed techniques can be employed for inference.

Fig. 5 illustrates the prediction mechanism using this approach. As training code is available, the user-developed code processes the training data, evaluates the model, and predicts the values. Multiple models can be swapped, hence making it modular, to determine the best-suited model for the application.

Fig. 6 introduces a generalized approach for predicting values using custom-designed techniques. Training code is deployed based on the training data, and techniques like hyperparameter tuning and TensorFlow/Keras can be employed. This approach allows for the integration and modeling of diverse production system models based on custom designs. The resulting models can be deployed on endpoints for real-time processing and serving.

Within the proposed modular AI architecture, TML is a crucial component. As a type of machine learning that focuses on structured, tabular data, TML is aptly equipped to handle the complexity and diversity of data found in many industrial processes, including testing systems. It efficiently manages high-dimensional data and intricate inter-variable relationships, making it well-suited for predicting outcomes of leak tests based on many parameters.

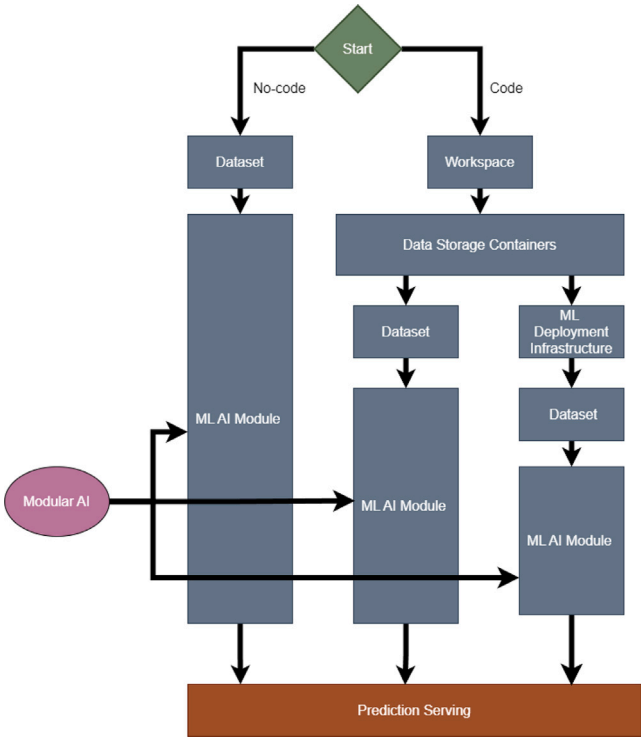

Fig. 6. Illustration of the mechanism for the prediction of values using the modular AI approach.

In the context of this modular AI approach, the incorporation of TML enhances the system's capability to learn from the rich, high-dimensional data generated in production systems. By doing so, it provides accurate predictions for leak test outcomes. The modularity

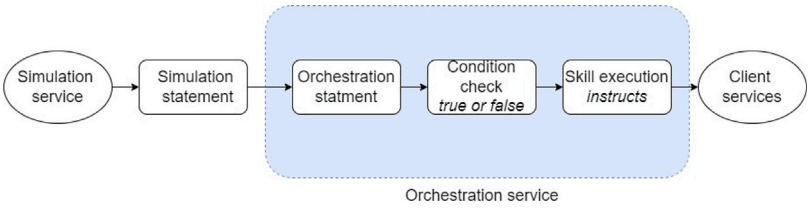

Fig. 7. Orchestration service description with its components.

of the AI system enables seamless swapping or upgrading of the TML model as required, thus offering a significant level of flexibility and adaptability. Therefore, the integration of TML into the modular AI system yields a robust and flexible tool for managing testing processes within production systems.

3.1.5. Orchestration service

The orchestration service guides the execution of skills on the asset through client services (Fig. 7). It receives the simulation results from the digital twin and transforms them into an orchestration statement. The statements are expressions that contain the information the services need for the execution of a skill. The process of developing these, the governing rules, and their behavior are explained in [32]. 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 is 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 testing.

3.1.6. 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 the skill submodel along with the pertaining data values from ML AI models (Fig. 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 behavior 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 ensures modularity and versatility in offering services.

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;

3.2.1. 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 information about the data on skills that could be executed, present in the manifest.

3.2.2. 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 the behavior requested by the orchestration service. Each client service is responsible for one skill per asset. These client services also contain information about the behavior of the particular skill. Fig. 8 presents an overview of the AAS integration architecture.

3.2.3. Value update

In test systems, the values need to be configured for specific products and conditional requirements. In the proposed approach, these values are determined and updated using modular AI deployment. The values are updated through the following steps:

- **Data Collection:** Data is collected from the production through the connected AAS. The captured information relates to the state of the system, functionality, and configuration. The connected AAS provides a digital footprint of the asset.
- **Digital Twin Simulation:** The digital twin receives the data from the physical asset along with the objective. The current Digital Twin is simulated to reach the objective, and the best action is formulated as an orchestration statement. The simulation environment can handle multiple data representations and supports optimization through software libraries.
- **Optimization Algorithms:** The simulation environment uses optimization algorithms to analyze the data and determine the production system's best configuration settings. These algorithms consider key performance indicators (KPIs) such as time and cost to infer the optimal configuration.
- **Communication and Integration:** The communication between the physical asset, digital representation, and optimization library is established using communication methods such as OPC UA, MQTT, or TCP/IP. This enables the exchange of data and configuration updates between the different components of the system.
- **Real-Time Configuration Update:** As the optimization algorithm selects the optimal configuration settings, the resulting configuration is loaded into the production system through APIs or PLC control.

The production system can continuously monitor its state, capture relevant data, simulate different configurations, optimize based on predefined KPIs, and update the configuration settings in real-time. This enables the system to adapt and self-configure according to changing requirements and optimize its performance based on the desired objectives.

3.2.4. Condition determination

In test systems, the condition of the state and configuration setting can be determined through information captured in AAS. Services can use these conditions to determine the possibilities of functionality execution. For instance, functionality depends on certain safety interlocks, i.e., states that can be queried through AAS. These conditions encapsulate the status quo of all components that make up the system.

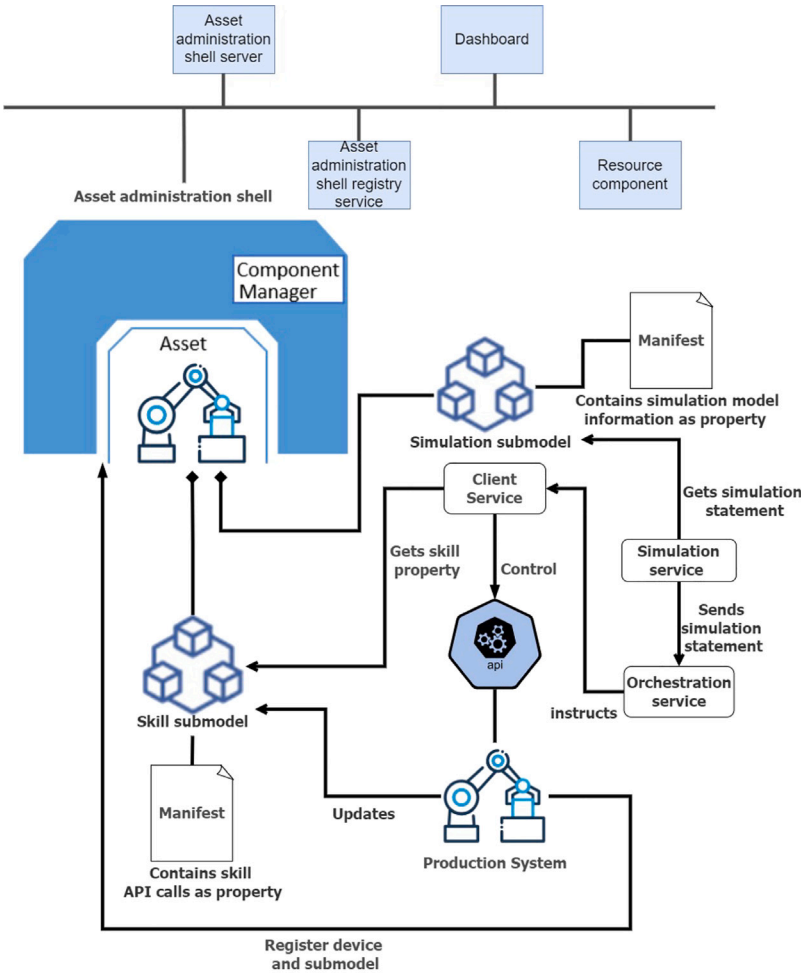

Fig. 8. Asset administration shell integration approach for testing processes. Control is guided through services, through skill and simulation submodels containing API calls and simulation data, respectively.

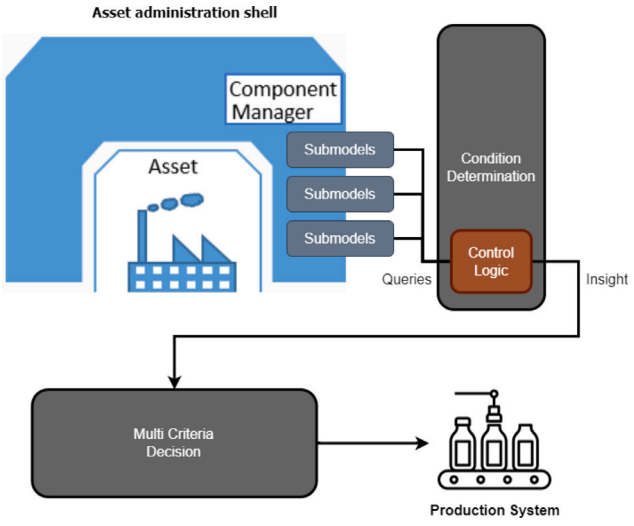

Fig. 9. Condition determination through AAS to ascertain status through control logic for decision-making on production systems.

Fig. 9 captures the condition-checking behavior, where the service queries the state through AAS. These services determine the status quo through control logic by referencing the properties of the submodel. This provides a means of formulating multi-criteria decision-making in production system execution. This approach presents an advantage over the traditional control approach, as a state can be queried without being referenced to I/O signals.

3.2.5. *Skill execution*

The client services execute skills through API calls retrieved from the skill submodel. System functionality is achieved through this execution. The AAS contains 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 the orchestration service proceeds to the next skill execution through a corresponding client service. Integrating 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 regularly to account for different requirements. These requirements can be a change in the 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 of the simulation service.

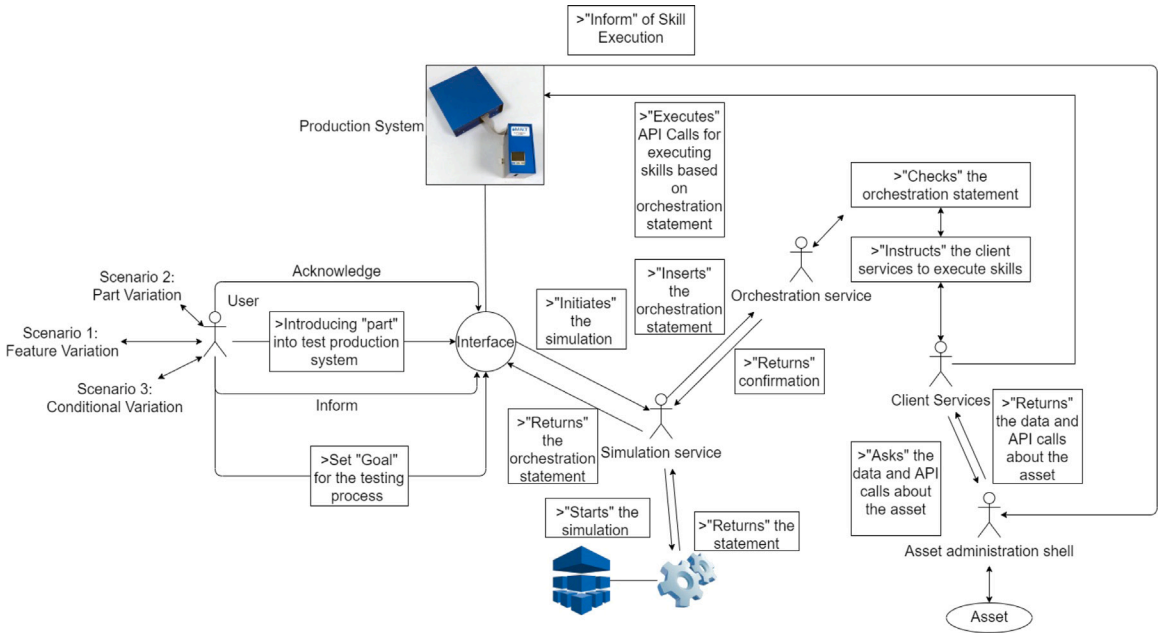

Fig. 10. Role of AAS integration approach in functionality execution with changing scenarios; part variation, feature variation, and conditional variation.

This approach can address scenarios such as part variation, feature, and conditional variation. Fig. 10 illustrates the AAS and the services in each scenario and the manner in which they coordinate for skill execution during operation.

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

4.1. Problem statement

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 Automation 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 [33] and AAS Explorer) is used 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 initialized by filling the part under observation with a testing medium or putting it in a state of vacuum. The part is stabilized over time under pressure or vacuum. The pressure change is measured and 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.

4.2. AAS representation for leak-test system

To represent the MALT leak test use case in the production system, capturing all the relevant information related to executing the leak test functionality is necessary. By using the AAS standard, we utilized a vendor-neutral approach to capturing this information. The AAS standard is utilized to capture the configuration specific to the leak test functionality in MALT (see Fig. 11).

The configuration for the leak test functionality is in terms of variables, their relationships, and constraints. These variables represent the configuration settings, while relationships are established through allocations and links. The AAS representation of the leak test functionality is linked to the production system (MALT) through an interface. As the AAS is updated, the connected production system is also updated via the interface, enabling dynamic configuration changes to be reflected in the MALT system.

The AAS representation for MALT consists of submodels (see Fig. 12) such as “Skill”, “Settings”, “Result”, and “Calibration”. The “Settings” submodel contains information about the parameters that need to be configured for executing the functionality, including their range and dependencies. The interface in MALT is capable of updating its configuration through each submodel, allowing for selective configuration updates without providing the entire AAS. However, strict structural requirements must be met for the submodel provided to the interface, as interfaces to the production systems adhere to strict structural requirements for submodels.

The AAS submodels provide detailed information for executing leak test functionality in the MALT production system. These submodels include “Calibration”, “Results”, and “Skill”. The “Calibration” submodel captures sensor calibration data. The “Results” submodel stores test results. The “Skill” submodel contains API calls and configuration data. This data is crucial for executing functionality. Thanks to the AAS representation, we can dynamically configure the leak test functionality. This ensures accurate configuration settings. It also provides a seamless information flow between the digital and physical systems.

4.3. Digital twin for leak-test system

The developed AAS functions as a digital representation of the physical asset, thereby serving as a digital twin of the leak-test system. For a leak-test execution, the associated services query the relevant information, as per product requirement, from the AAS submodels. Fig. 13 provides an illustration of this process.

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 a simulation statement from the simulation service is:

$$Simulation\ statement = \langle basic : P_{ML} :$$

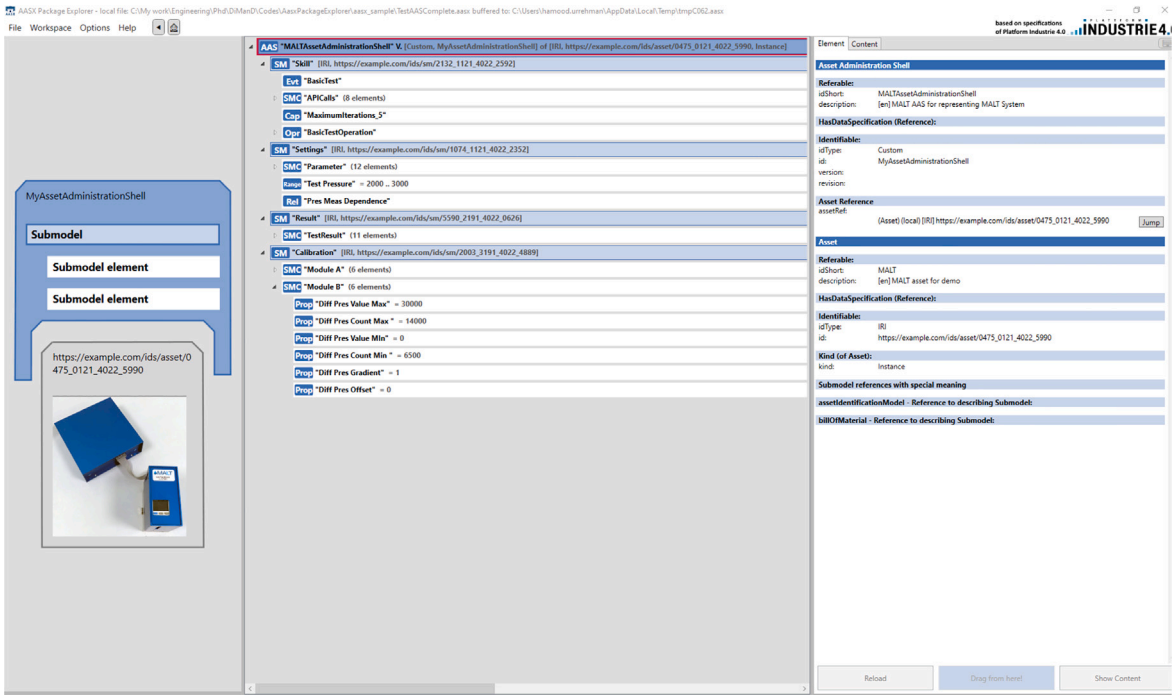

Fig. 11. AAS for MALT.

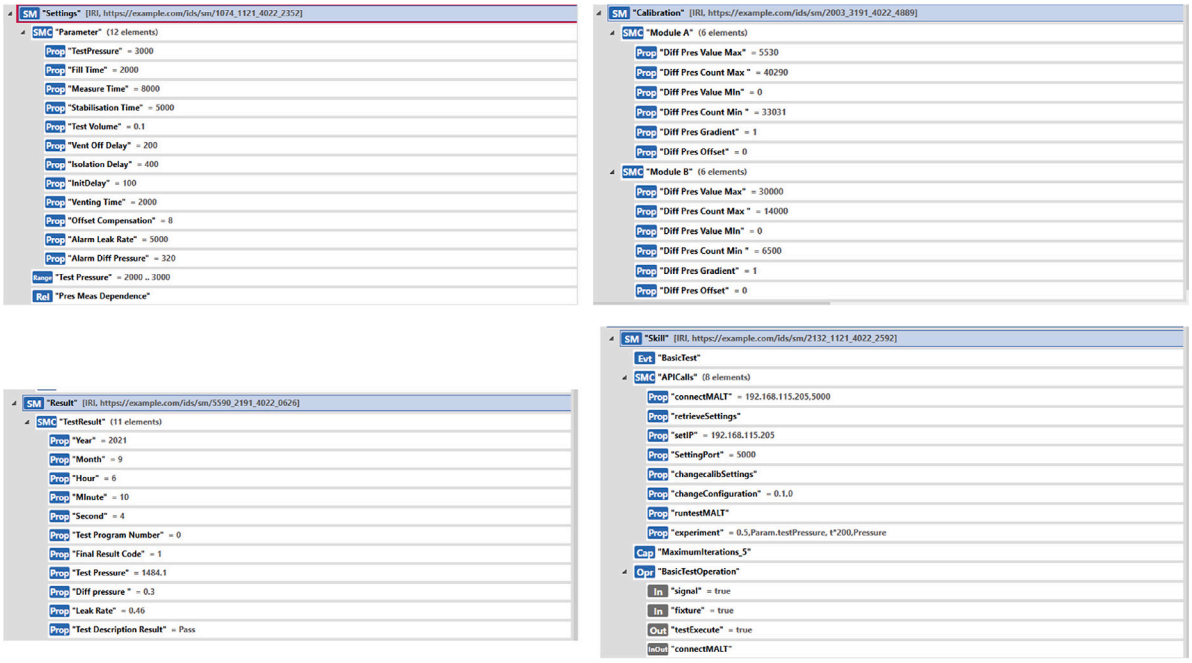

Fig. 12. Detail of each submodel to execute "leak test" functionality.

$$P_{ML}\rangle.\langle condition : MaxTestPressure\ P_{ML}\rangle.$$
$$\langle stab : P_{ML}\rangle.\langle fill : P_{ML}\rangle.$$
$$\langle param : measure : P_{ML}\rangle.\langle execute\rangle \quad (1)$$

In leak testing, we provide the goal to "execute" a test. Conditions were provided for the simulation. The sequence of execution is followed by the combination passed from the simulation statement. The parameter (P_{ML}) for the simulation statement is determined from the machine learning application and then simulated for the leak test case. The parameters are updated in the system for execution.

4.4. Execution

The orchestration statement acts as a soft controller, guiding the execution of skills from one state to another.

$$Orchestration\ Statement = [basic].[condition].$$
$$[stab].[fill].[param].[execute] \quad (2)$$

The "basic" client service starts the connection with the testing system, the "condition" service uses conditions to change the testing parameters, and stabilization time is adjusted through the "stab" client service. The "fill" and "param" client services change the fill time

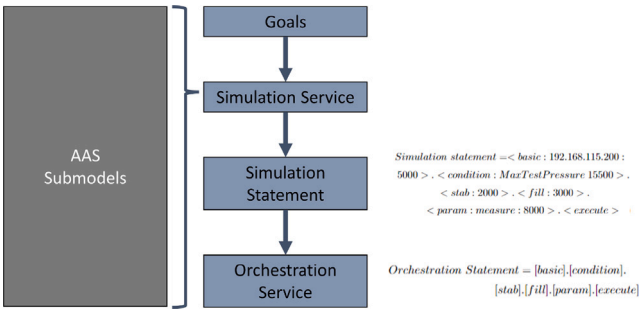

Fig. 13. Simulation service querying information from AAS to determine simulation statement based on goals from user and product information. The Machine Learning application determines the parameters.

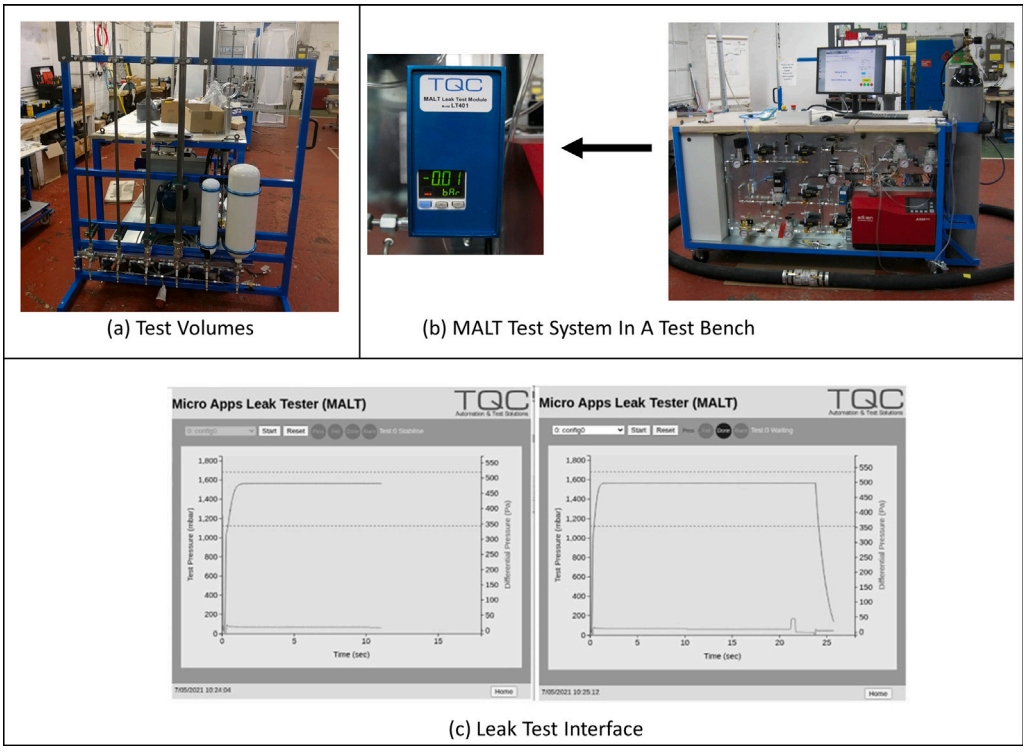

Fig. 14. The leak testing setup: (a) the cylinder volumes under testing (b) MALT testing system, a part of the test bench for general leak testing (c) Interface for leak testing; asset administration shell drives the execution through client services.

and any parameters (taken as arguments), 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 information on the manner in which each skill for leak testing needs to be executed. The client service takes in leak testing data from the AAS submodel and produces changes as desired. The desired change in settings by each client service is loaded through an API call from the AAS skill submodel, and 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 the testing process is given in Fig. 14.

This approach provides a control mechanism for executing testing processes controlled through services. The services utilize the Type 2 AAS representation to produce behavioral 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, it makes the approach modular and cost-effective as multiple individual component requirements need not be addressed.

4.5. Modular machine learning application

The machine learning application for this research is developed using a modular approach, which entails breaking down the application into separate, self-contained components. The application is structured into several modules, each responsible for a specific aspect of the process: *Data Preparation*, *Model Training and Testing*, and *Visualization*. This modular architecture conforms to the principle of separation of concerns, allowing each module to carry out a distinct function independent of the others. Such a design is not only efficient but also improves the maintainability and understandability of the system.

A vital role of this modular machine learning application is to determine the parameters used by simulation services to assess the system functionality before other services update these parameters onto the leak test system. The queried parameters' values are derived from the AAS submodels and processed by our machine-learning application. This modular approach permits easy modification and expansion, promoting adaptability to evolving research requirements.

This application aims to train a model capable of predicting leak test outcomes, categorized into four classes: 0 (Fail - Test Pressure High), 1 (Pass), 2 (Fail - Test Pressure Low), and 3 (Fail).

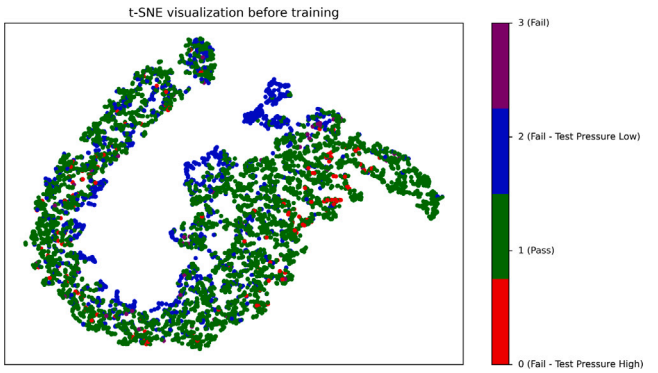

Fig. 15. Visualization of the original dataset using the t-SNE method.

4.5.1. Data preparation

The first module of the machine learning application is the data preparation module. In many diverse fields, such as finance, marketing, healthcare, and more, tabular data is the most common form of data used for machine learning. It comprises structured data that is made up of rows and columns, similar to a spreadsheet or a relational database. Each row of the table represents a single instance, and each column in the table represents a feature or attribute of the instance. The data is usually a mixture of numerical and categorical variables.

The model's training uses a dataset (which is transferred already into a tabular form) from the MALT at TQC, comprising 9076 unique configurations of leak tests. Each configuration is defined by 12 parameters: Volume, Initialisation Delay, Test Pressure (mbar), Vent Off Delay, Fill Time, Stabilisation Time, Isolation Delay, Measuring Time, Venting Time, Offset Compensation, Alarm Leak Rate (mm^3/s or mbar/s), and Alarm Differential Pressure (Pa or mbar). Each configuration is associated with an outcome belonging to one of the four classes. These features constitute the inputs for our machine learning model. Modular data preparation allows for handling diverse data sources and types, making it adaptable to various scenarios.

To aid the training and evaluation process, the dataset is divided into three subsets: a training set (64% of the total data), a validation set (16% of the data), and a test set (the remaining 20%).

However, with high-dimensional data, understanding these relationships can be challenging. For this reason, a two-dimensional representation of the dataset is created using the t-distributed Stochastic Neighbor Embedding (t-SNE) method [34] for visualization purposes, as shown in Fig. 15. This visualization highlights the necessity of applying machine learning methods for effective classification.

4.5.2. Model training and testing

The model training process is encapsulated in the machine learning application as a distinct module. This modular approach allows for the machine learning model's independent development, testing, and optimization. The module takes the preprocessed tabular data as input and is responsible for training the machine learning model.

The implementation of a Multilayer Perceptron (MLP) model [35] for the classification task has yielded significant results. An early stopping mechanism [36] is employed during training to prevent overfitting and to improve computational efficiency. This mechanism halted the training at the 348th epoch, as further training did not show improvement in the validation accuracy.

The performance of the MLP model is depicted in Fig. 16, which shows a consistent reduction in training loss, reaching a low point of 0.0275. This decrease in loss indicates the model's increasing proficiency in predicting the correct outcomes over the training epochs. Alongside the decrease in loss, the model's accuracy, representing the proportion of correct predictions, also showed considerable improvement. As illustrated in Fig. 17, the training accuracy of the model peaked at an impressive 99.31%, demonstrating its effectiveness in the classification task.

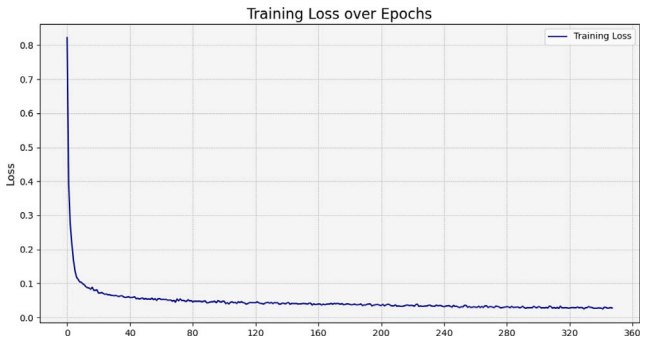

Fig. 16. Training loss over epochs.

Fig. 17. Training accuracy over epochs.

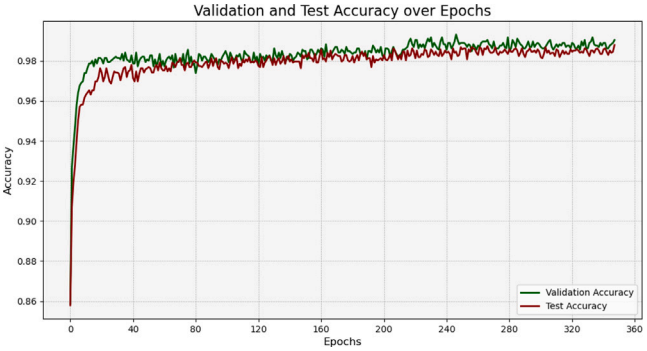

Fig. 18. Validation and test accuracy over epochs.

Additionally, the performance of the model on unseen data, as measured by validation and test accuracy, is presented in Fig. 18. These metrics provide insights into the model's generalization capabilities beyond the training data.

4.5.3. Visualization

The t-SNE is employed again post-training to visualize the transformed dataset with the trained machine-learning model. As described in Section 4.5.1, this technique is crucial for interpreting the high-dimensional outputs of the model within a two-dimensional space, particularly considering the 12 attributes involved. The effectiveness of t-SNE lies in its ability to illustrate the clustering or separation of data points after being processed by the model.

As depicted in Fig. 19, the t-SNE visualization post-training distinctly separates the four classes corresponding to the leak test outcomes. This separation is a clear indication of the model's capability in

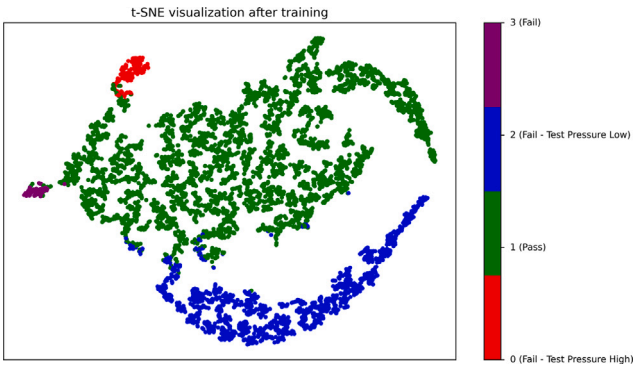

Fig. 19. T-SNE visualization of the dataset after model training, demonstrating the effective classification of leak test results.

accurately classifying each leak test configuration based on the defined attributes.

In practical applications, this visualization demonstrates that the trained machine learning model is capable of predicting the outcome of new leak test configurations. These outcomes are classified into four distinct categories: 0 (Fail - Test Pressure High), 1 (Pass), 2 (Fail - Test Pressure Low), and 3 (Fail). The clarity in class separation shown in the t-SNE plot post-training underscores the model's effectiveness in such predictions.

In conclusion, the modular machine learning approach implemented in this research offers numerous benefits. It provides the flexibility to adapt to new requirements and changes, enhances maintainability by facilitating understanding and debugging, and improves scalability, making it suitable for large-scale applications. Each module in the pipeline, from data loading and preprocessing to model training and performance evaluation, can be developed, tested, and optimized independently, catering to a wide range of datasets and tasks. The modular approach aligns with the future direction of machine learning, offering a promising solution for complex and large-scale applications.

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 digitalization 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 maximize impact, especially for SMEs. AAS is the coming standard for digitalizing manufacturing assets and maximizing modularity and interoperability. A static AAS is inadequate for adapting to testing needs; type 2 is required.

This research showcases an application of a modular AI approach integrated with Type 2 AAS for testing processes. The modular AI design offers significant benefits, as it enables the separation of different components or modules in the system, each responsible for specific functionality. Such an architecture greatly enhances the system's flexibility and scalability. It allows for easy adjustments and modifications of individual modules without disrupting the entire system, which can be particularly beneficial in adapting to evolving testing requirements. Furthermore, the modular approach facilitates the maintenance and troubleshooting processes, as issues can be isolated and addressed at the module level.

This research incorporates tabular machine learning into a modular AI system framework. Tabular machine learning is well-suited for dealing with high-dimensional data. It also facilitates an understanding of complex inter-dependencies among the multiple parameters involved in the testing process. This ability contributes to the precise prediction of leak test outcomes, thereby improving the reliability and effectiveness of the testing process.

The integrated approach presented in this research contributes significantly towards managing the functionality of testing processes within production systems. It drastically reduces the setup time for a new testing process, thereby accelerating the production cycle and enabling faster time-to-market for products. Moreover, it minimizes the need for continuous expert intervention to control the testing operation, leading to substantial cost savings and efficient resource allocation.

In essence, the application of a modular AI approach, integrated with Type 2 AAS and enhanced with the power of tabular machine learning, offers businesses an efficient, cost-effective, and adaptable solution for managing their testing operations. This can lead to improved operational efficiency, cost savings, and, ultimately, enhanced business competitiveness.

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.

In addition to the points discussed earlier, it is essential to emphasize the particular advantages of a modular architecture in managing large, complex, and continually evolving processes. The modular design of AI systems, as demonstrated in this research, offers unparalleled flexibility and adaptability. This is especially crucial in dynamic industries where testing requirements and processes frequently change. By adopting a modular approach, each component of the system can be independently modified, enhanced, or replaced without impacting the overall system's integrity. This not only ensures that the system remains up-to-date with the latest technological advancements but also allows it to adapt swiftly to new or evolving requirements.

Moreover, the modular architecture significantly simplifies the integration of new components and technologies. As the demands of testing processes evolve, new modules can be seamlessly incorporated into the existing system, ensuring that the system remains at the forefront of technological efficiency. This approach also facilitates easier scaling, making it suitable for both small-scale operations and large-scale industrial applications.

In light of these benefits, future work will delve deeper into demonstrating the efficacy of modular architecture in more complex and larger-scale scenarios. The forthcoming research will focus on illustrating how this approach can effectively manage the intricacies and challenges of more substantial and complex testing processes, further solidifying the argument for its widespread adoption in industrial settings.

Other future work 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 through swappable ML AI models. Service negotiation behavior and transition approach from Type 2 AAS to Type 3 AAS will be explored. As seen in the application in this research, AAS has the potential towards optimizing industrial processes, workflows, and resource utilizations. Its data-driven nature promotes adaptability and integration with external services. More research will be carried out in AAS with a focus on automation to bring down costs and promote data-driven decision-making in the industry.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research is supported by the DiManD Innovative Training Network (ITN) project funded by the European Union through the Marie Skłodowska-Curie Innovative Training Networks (H2020-MSCA-ITN-2018) under grant agreement number no. 814078.

References

[1] Kostal P, Mudrikova A, Michal D. Possibilities of intelligent flexible manufacturing systems. In: IOP conference series: Materials science and engineering, Vol. 659. IOP Publishing; 2019, 012035.
[2] Yang Bo, Pang Zhi, Wang Shilong, Mo Fan, Gao Yifan. A coupling optimization method of production scheduling and computation offloading for intelligent workshops with cloud-edge-terminal architecture. J Manuf Syst 2022;65:421–38.
[3] Pulikottil Terrin, Estrada-Jimenez Luis A, Abadía José Joaquín Peralta, Carrera-Rivera Angela, Torayev Agajan, Rehman Hamood Ur, Mo Fan, Nikghadam-Hojjati Sanaz, Barata Jose. Big data life cycle in shop-floor–trends and challenges. IEEE Access 2023.
[4] Mo Fan, Chaplin Jack C, Sanderson David, Martínez-Arellano Giovanna, Ratchev Svetan. Semantic models and knowledge graphs as manufacturing system reconfiguration enablers. Robotics and Computer-Integrated Manufacturing 2024;86:102625.
[5] Sharp Michael, Ak Ronay, Hedberg Jr Thomas. A survey of the advancing use and development of machine learning in smart manufacturing. J Manuf Syst 2018;48:170–9.
[6] Aggour Kareem S, Gupta Vipul K, Ruscitto Daniel, Ajdelsztajn Leonardo, Bian Xiao, Brosnan Kristen H, Kumar Natarajan Chennimalai, Dheeradhada Voramon, Hanlon Timothy, Iyer Naresh, et al. Artificial intelligence/machine learning in manufacturing and inspection: A GE perspective. MRS Bull 2019;44(7):545–58.
[7] Rehman Hamood Ur, Pulikottil Terrin, Estrada-Jimenez Luis Alberto, Mo Fan, Chaplin Jack C, Barata Jose, Ratchev Svetan. Cloud based decision making for multi-agent production systems. In: Progress in artificial intelligence: 20th EPIA conference on artificial intelligence, EPIA 2021, Virtual event, September 7–9, 2021, Proceedings 20. Springer; 2021, p. 673–86.
[8] McFarlane Duncan, Sarma Sanjay, Chirn Jin Lung, Wong ChienYaw, Ashton Kevin. Auto ID systems and intelligent manufacturing control. Eng Appl Artif Intell 2003;16(4):365–76.
[9] Zhong Ray Y, Xu Xun, Klotz Eberhard, Newman Stephen T. Intelligent manufacturing in the context of industry 4.0: a review. Engineering 2017;3(5):616–30.
[10] Mo Fan, Rehman Hamood Ur, Elshafei Basem, Chaplin Jack C, Sanderson David, Martínez-Arellano Giovanna, Ratchev Svetan. Efficient decision-making in SMEs: Leveraging knowledge graphs with neo4j and AI vision. In: 1st workshop on low-cost digital solutions for industrial automation at the University of Cambridge. 2023.
[11] Rehman Hamood Ur, Chaplin Jack C, Zarzycki Leszek, Ratchev Svetan. A framework for self-configuration in manufacturing production systems. In: Doctoral conference on computing, electrical and industrial systems. Springer; 2021, p. 71–9.
[12] Mo Fan, Querejeta Miriam Ugarte, Hellewell Joseph, Rehman Hamood Ur, Rezabal Miren Illarramendi, Chaplin Jack C, Sanderson David, Ratchev Svetan. PLC orchestration automation to enhance human–machine integration in adaptive manufacturing systems. J Manuf Syst 2023;71:172–87.
[13] Schweichhart Karsten. Reference architectural model industrie 4.0 (rami 4.0). 2016, An Introduction. Available online: https://www.plattform-i40.deI. 40.
[14] Wei Kang, Sun JZ, Liu RJ. A review of asset administration shell. In: 2019 IEEE international conference on industrial engineering and engineering management (IEEM). IEEE; 2019, p. 1460–5.
[15] Rehman Hamood Ur, Chaplin Jack C, Zarzycki Leszek, Mo Fan, Jones Mark, Ratchev Svetan. Service based approach to asset administration shell for controlling testing processes in manufacturing. IFAC-PapersOnLine 2022;55(10):1852–7.
[16] Wenger Monika, Zoitl Alois, Müller Thorsten. Connecting PLCs with their asset administration shell for automatic device configuration. In: 2018 IEEE 16th international conference on industrial informatics (INDIN). IEEE; 2018, p. 74–9.
[17] Bedenbender H, Billmann M, Epple U, Hadlich T, Hankel M, Heidel R, Hillermeier O, Hoffmeister M, Huhle H, Jochem M, et al. Examples of the asset administration shell for Industrie 4.0 components–basic part. ZVEI white paper, 2017.
[18] Wagner Constantin, Grothoff Julian, Epple Ulrich, Drath Rainer, Malakuti Somayeh, Grüner Sten, Hoffmeister Michael, Zimermann Patrick. 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). IEEE; 2017, p. 1–8.
[19] Cavalieri Salvatore, Salafia Marco Giuseppe. Insights into mapping solutions based on opc ua information model applied to the industry 4.0 asset administration shell. Computers 2020;9(2):28.
[20] Cavalieri Salvatore, Salafia Marco Giuseppe. Asset administration shell for PLC representation based on IEC 61131–3. IEEE Access 2020;8:142606–21.
[21] Tantik Erdal, Anderl Reiner. Potentials of the asset administration shell of Industrie 4.0 for service-oriented business models. Procedia CIRP 2017;64:363–8.
[22] Pethig Florian, Niggemann Oliver, Walter Armin. Towards Industrie 4.0 compliant configuration of condition monitoring services. In: 2017 Ieee 15th international conference on industrial informatics (Indin). IEEE; 2017, p. 271–6.
[23] Löcklin Andreas, Vietz Hannes, White Dustin, Ruppert Tamás, Jazdi Nasser, Weyrich Michael. Data administration shell for data-science-driven development. Procedia CIRP 2021;100:115–20.
[24] Mo Fan, Rehman Hamood Ur, Monetti Fabio Marco, Chaplin Jack C, Sanderson David, Popov Atanas, Maffei Antonio, Ratchev Svetan. A framework for manufacturing system reconfiguration and optimisation utilising digital twins and modular artificial intelligence. Robot Comput-Integr Manuf 2023;82:102524.
[25] Quarantiello Luigi, Marzeddu Simone, Guzzi Antonio, Lomonaco Vincenzo. LuckyMera: a modular AI framework for building hybrid NetHack agents. 2023, arXiv preprint arXiv:2307.08532.
[26] Qiao Litao, Wang Weijia, Dasgupta Sanjoy, Lin Bill. Rethinking logic minimization for tabular machine learning. IEEE Trans Artif Intell 2022.
[27] Shwartz-Ziv Ravid, Armon Amitai. Tabular data: Deep learning is not all you need. Inf Fusion 2022;81:84–90.
[28] Westreich Daniel, Lessler Justin, Funk Michele Jonsson. Propensity score estimation: machine learning and classification methods as alternatives to logistic regression. J Clin Epidemiol 2010;63(8):826.
[29] Grinsztajn Léo, Oyallon Edouard, Varoquaux Gaël. Why do tree-based models still outperform deep learning on typical tabular data? Adv Neural Inf Process Syst 2022;35:507–20.
[30] Natekin Alexey, Knoll Alois. Gradient boosting machines, a tutorial. Front Neurorobot 2013;7:21.
[31] Talukder Md Alamin, Islam Md Manowarul, Uddin Md Ashraf, Akhter Arnisha, Hasan Khondokar Fida, Moni Mohammad Ali. Machine learning-based lung and colon cancer detection using deep feature extraction and ensemble learning. Expert Syst Appl 2022;205:117695.
[32] Rehman Hamood Ur, Chaplin Jack C, Zarzycki Leszek, Jones Mark, Ratchev Svetan. Application of multi agent systems for leak testing. In: 2021 9th international conference on systems and control (ICSC). IEEE; 2021, p. 560–5.
[33] BaSyx Eclipse. Eclipse BaSyx. Enabling Open Innov Collab 2022.
[34] Linderman George C, Rachh Manas, Hoskins Jeremy G, Steinerberger Stefan, Kluger Yuval. Efficient algorithms for t-distributed stochastic neighborhood embedding. 2017, arXiv preprint arXiv:1712.09005.
[35] Taud Hind, Mas JF. Multilayer perceptron (MLP). In: Geomatic approaches for modeling land change scenarios. Springer; 2018, p. 451–5.
[36] Sitaula Chiranjibi, Ghimire Nabin. An analysis of early stopping and dropout regularization in deep learning. Int J Concept Comput Inf Technol 2017;5(1):17–20.