Deutscher Luft- und Raumfahrtkongress 2023
DocumentID: 610125

MASIMO – DEVELOPMENT AND RESEARCH OF INDUSTRY 4.0 COMPONENTS WITH A FOCUS ON EXPERIMENTAL APPLICATIONS OF PROACTIVE ASSET ADMINISTRATION SHELLS IN DATA-DRIVEN MAINTENANCE ENVIRONMENTS

M. Weiss[+,*], K. Wicke[+], G. Wende[+], H. Pakala[**], M. S. Gill[‡]
[+] Deutsches Zentrum für Luft- und Raumfahrt, DLR-MO, Hamburg, Germany
[**] Otto von Guericke Universität, OvGU-LIA, Magdeburg, Germany
[‡] Helmut-Schmidt-Universität, HSU-AUT, Hamburg, Germany

Abstract

In the research agendas of the EU, the USA and many countries on the digitalization of industry and society (e.g. [5], [11], [14], [31]), the creation of an Internet of Things (IoT) with its digital representations of assets is described as one of the most important transformations for future life and work. Many of the corresponding activities are summarized under the term Industry 4.0 (I4.0). The paper discusses how data-driven MRO can benefit from I4.0 developments, in particular by applying I4.0 components, which are the combination of an asset and its digital complement, the Asset Administration Shell. The paper proposes a data-centric MRO environment through the creation of a cyber-physical system for seamless and secure communication and smart interoperation between all stakeholders - such as humans, machines, or tangible and intangible products. In a conceptual design stage, the mandatory elements are introduced, such as independently operating proactive AAS, the necessary hardware adapters or the descriptive submodels of AAS with regard to the requirements of the MRO. The interactions between the AAS are based on their bidding capabilities, including the automated exchange and negotiation of tender documents, for which a structure is proposed. In an implementation scenario, the introduced concept is experimentally verified.

1. INTRODUCTION

In the context of the manufacturing-oriented Industry 4.0 (I4.0) activities, several architectures, concepts, and solutions have emerged that are also applicable to future data-driven maintenance, as we discuss in [38]. A key element is the Asset Administration Shell (AAS) standardized 2022 [12], which promises seamless interoperation, structured data management, and intelligent decision making for various assets in cyber-physical-social systems (CPSS), as we present in [39]. The goal is to build a cross-domain, cross-stakeholder and cross-technology digitized MRO value chain that is in line with the I4.0 vision of an open IoT marketplace (Figure 1). Here, all participants (from companies to individuals) exchange their service requests and offers with each other or maintain their lifecycle data using the AAS as a common communication framework.

The AAS is described by the Plattform Industrie 4.0 as a digital representation of an asset and is also referred to as a "digital twin", but can go significantly beyond this due to its intended proactive behavior. It is this proactive behavior that enables self-determined communications and negotiations between assets in scalable CPSS (from the field level to global networking) regarding specific requests and responses as expected capabilities/skills to meet desired demands. Based on the overarching research question [38],

RQ 1 To what extent can the standards and recommendations of Industrie 4.0 Components be applied in MRO, including humans?

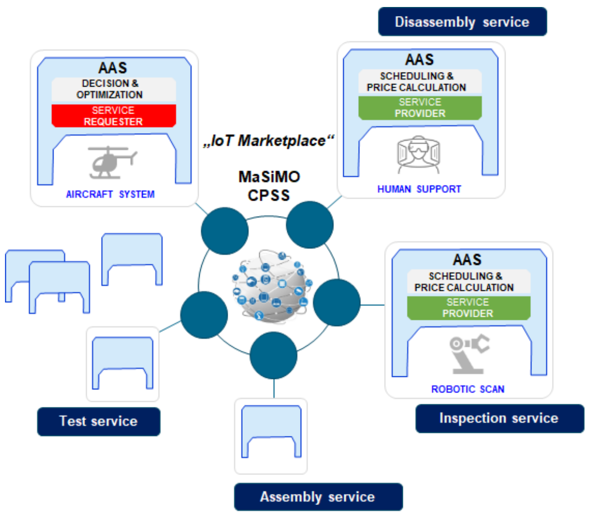


Figure 1: Interoperation in the CPSS, based on [4]

the deeper focus in the present paper is on elaborating the question,

RQ 2 How can the Asset Administration Shell concept be used, adapted or extended to be applied in data-driven MRO in the aviation sector?

Building on our research results presented in [38] and [39] and considering the latest developments of I4.0, the paper will describe the preliminary design and construction of I4.0 components (synthesis of assets and their administration

*) *Corresponding author: Marco Weiss – Marco.Weiss@DLR.de*

doi: 10.25967/610125

shell) for a data-driven MRO system. For this purpose, the necessary software and hardware components for an HMI (human-machine interface), a robot, and a drone as an aircraft system surrogate are presented – all of which can be MRO service requester or providers. Complementary, first concepts for maintenance-based submodels (SM; mandatory descriptive parts in an AAS) for autonomous negotiations between the service requesters and providers regarding maintenance tasks are introduced. Among other submodels, there is a particular focus on seamless communication (including all information required for the request and offer in a bidding process, such as skills, availability, costs, desired location etc.) between all actors involved in the CPSS, such as the humans, the tools, and the service-receiving assets. The exchange will be based on the VDI 2193 standard ([34], [35]) for structured messages and interaction protocols as a language for I4.0 components.

2. INTEROPERATION OF INTANGIBLE AND TANGIBLE ASSETS VIA AAS IN A CPSS

In [38], we propose the fundamentals of a cyber-physical-social system of a data-driven maintenance environment. It consists of data networks and Asset Administration Shells (AAS) as interoperating components. The Plattform Industrie 4.0 [11] defines the AAS as a standardized digital representation of an asset ("digital twin") to make it available in digital data spaces (DIN EN IEC 63278-1 [12]). Thus, the PI4.0 introduced a meta-model to digitally represent an asset in an anticipation to give uniform access to information and behavior of the asset across companies and thereby addressing the aspect of interoperability to some extent [7]. Since 2021, the Industrial Digital Twin Association (IDTA) has been developing, integrating and disseminating the AAS PI4.0 concept to bring it to maturity. The IDTA regularly publishes standards and guidelines for the AAS architecture and its submodels[1]. The latter normatively describe content-related (e.g. identification, documents, certificates) or functional aspects (e.g. capabilities and skills) of an AAS. The submodels are part of the AASX file, a generic package file format that contains the Asset Administration Shell structure, data and other related files [19]. The AASX file supports the seamless exchange of data between organisations/partners and the storage/consistency of Asset Administration Shell information. In short, it contains and organizes all the information required for a digital representation of an asset.

There are three types of the AAS (Figure 2): the digital representation in the form of a file that can be exchanged between trading partners is defined as the Type 1 AAS. PI4.0 describes **reactive AAS** as Type 2, which have an API (e.g. REST) through which the digital representation of the asset can exchange data on request. The ability to work autonomously in a manufacturing or maintenance environment, e.g. to make independent decisions, is attributed to Type 3 AAS - **proactive AAS**.

Different participants in the CPSS may have different views of the asset and thus different expectations of its role and capabilities of the AAS in general. In some cases, the pure exchange of digital data about the asset is sufficient (e.g. nameplate, lifecycle data, CAD files); in other cases, the asset's autonomous behavior as a self-managed service requester (SR) or service provider (SP) is expected. These different roles are considered through customized submodels based on normative templates where possible. Additionally, the expected role decides about the type of the AAS. In our research, the proactive AAS was selected because it can be designed to interact autonomously with other AAS in the CPSS: using built-in algorithms, it makes decisions and negotiates the request or provision of capabilities and skills in bidding processes. Thus, we focus on submodels that support the implementation of capabilities and skills for self-determined interactions. "In this context, a capability is defined as an '*implementation-independent specification of a function'* [27], whereas a skill is defined to be the executable implementation of said function" [37].

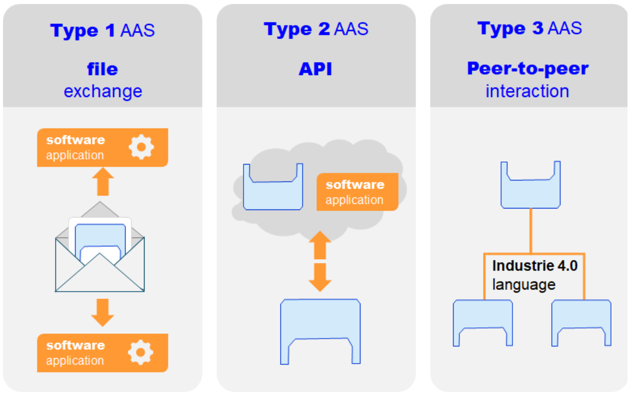


Figure 2: Types of AAS, based on [19]

The proactive AAS interprets and operationalizes the submodels in order to manage the behavior of its associated asset at a kind of top level (Figure 3). The behavior is modelled with specific subcomponents using explicit methods or finite state machines, which in turn interact with the asset itself (e.g. write or read parameters via OPC UA) and thus encapsulating complex functions of the asset. In combination with the asset, a proactive AAS can be viewed as an I4.0 component whose subcomponents model characteristic features, such as a decision algorithm that triggers certain events. This, combined with the ability of self-managed interactions between participants in the CPSS environment, makes the proactive AAS unique in a smart, data-driven network of interoperable, cross-domain and cross-technology assets.

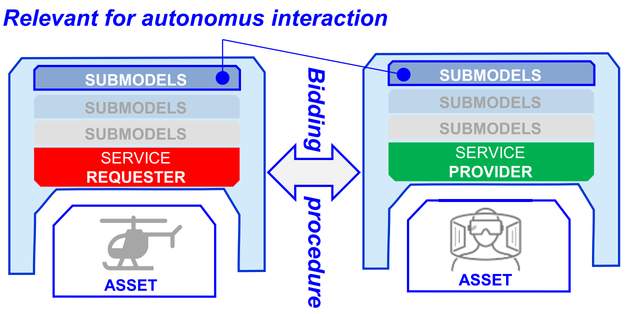


Figure 3: Selection of SM for proactive AAS, based on [1]

[16] describes the interoperation of proactive AAS with its environment using interaction protocols (IP) that use the I4.0 language format. The IP is a structured sequence of interactions for the exchange of I4.0 messages between I4.0 components. An I4.0 message as defined in the VDE 2193-Part 1 [34] consists of interactive elements and a

[1] https://industrialdigitaltwin.org/en/content-hub/submodels (02.08.2023)

frame: The frame comprises sender and receiver information, message type, conversation and messages IDs and a reference to the used protocol. The interactive elements comprise submodels or other data elements that are exchanged between the AAS. Following [16], [2] advances the concept of I4.0 component interaction as an environment of interacting state machines, with an example application is shown in [1]. In this context, the interaction protocol of an iterative bidding procedure between I4.0 components (VDE 2193-Part 2, [35]) is of great importance, as will be shown later.

2.1. Software and hardware of an I4.0 Component with type 3 (proactive) AAS

In preparation for a standardized implementation of the assets in a CPSS, one idea here is to provide a domain-independent and easy-to-implement hardware box that builds an I4.0 component together with the asset. The box is intended to be used as an uniform plug-in device for a wide variety of assets and to provide them with additional features if they are not available by default (e.g. PLC with built-in OPC UA server). The aim is also to improve already existing infrastructures without great effort (brownfield scenario).

First, it includes the functionality of reading, writing, processing, and protocol-based access to the asset's data variables (Figure 4). Second, these data variables and thus the asset are managed by encapsulating them with a Python-based proactive AAS and its smart behavior (Figure 5). The latter enables the asset to make self-determined decisions based on its data (e.g. health condition) or in the bi-directional bidding procedures between other participants of the CPSS as a service requester (customer) or provider (contractor; chapter 2.2.1). The box is connected to the asset via serial, loopback or Ethernet interfaces and to the CPSS via Ethernet, WLAN or LTE interfaces.

2.1.1. Software development

Data server: The very first step is to make the asset’s bi-directional data stream available to the proactive AAS and thus to the CPSS network. To do this, a data server is created. This is based on a data integrator in combination with communication adapters. First, all data variables of the asset required for its initializing, monitoring and control have to be exchanged, prepared or even processed (e.g. translation of raw data) before being transmitted to the network communicators and vice versa. This is done via the data integrator (Figure 4, top), which is programmed in Python, connected to the asset (serial, Ethernet) and continuously exchanging the raw or pre-processed data with the following two network communication adapters (Figure 4, center):

The first option is to use an OPC UA server[2], which provides network accessible variables for data exchange via its own protocol. By configuring the OPC UA server with appropriate endpoint settings such as URL and namespace, clients like the AAS server (see below) can connect to the data server. Additionally, an event handler is implemented to detect changes in the data stream of the asset and trigger corresponding updates in the OPC UA server. This event-driven approach ensures that the OPC UA server is continuously informed about any modifications in the data, allowing for real-time synchronization between the OPC UA Server and the connected clients (e.g. AAS; Figure 4 bottom). The second option is to use a REST server built with Flask[3], which offers a REST API via endpoint definitions that provides a range of operations such as retrieving, adding, updating or deleting data. Communication with clients is done using HTTP methods and responses are returned in JSON format. Before the proactive AAS can manage the asset, the data flow between the two must be established in a server-client combination. Here we use OPC UA as a common exchange format.

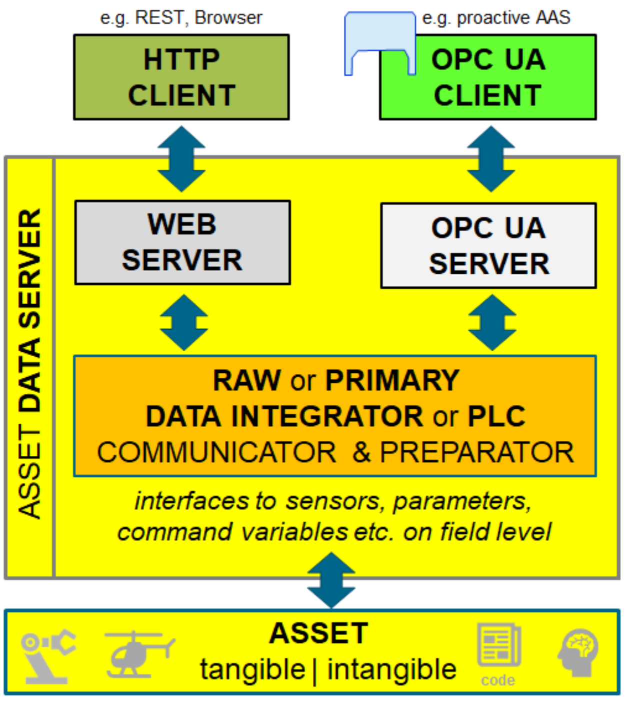

e.g. REST, Browser
HTTP CLIENT
e.g. proactive AAS
OPC UA CLIENT
ASSET DATA SERVER
WEB SERVER
OPC UA SERVER
RAW or PRIMARY DATA INTEGRATOR or PLC COMMUNICATOR & PREPARATOR
interfaces to sensors, parameters, command variables etc. on field level
ASSET
tangible | intangible
code

Figure 4: Scheme of data server, managing bi-directional data flows between tangible or intangible assets and standardized interfaces and communication protocols (HTTP, OPC UA)

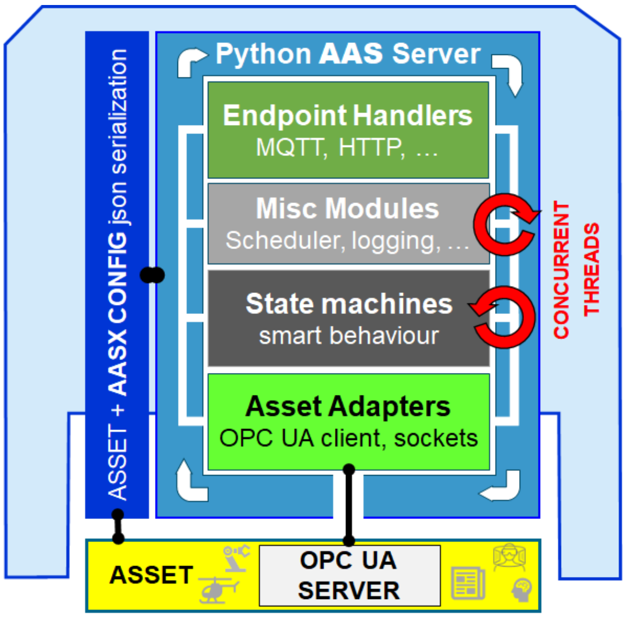
AAS server: In [38], chapter 3.2, we describe a concept of how to design a proactive AAS and thus a necessary system architecture could look like. In the present work, the concept is a merged result of different proposals on this topic, in particular by [2], [23] and [24]. Here, the findings were applied to design a Python-based solution in terms of the previously referenced architecture.

In general, and as introduced in e.g. [2], [32] and [36], a **component manager** is responsible for the overall orchestration of the datasets and algorithms that control data processing and dynamic behavior of the AAS. In the Python-based AAS, this is realized by a centralized orchestrating server class as the base module on which a whole package of modules is built (Figure 5). The modules cover several tasks, the most important of which are initializing the AASX file, maintaining the AASX data structure in memory, operating an internal data bus, handling incoming and outgoing I4.0 messages via events and forwarding them through the system, or providing various communication adapters (REST, OPC UA, MQTT, HTTP). In addition, another module implements a set of state machines that support the AAS to behave independently, with a focus on interaction between multiple AASs in the CPSS. The module resembles the **interaction manager** as recommended for

[2] https://github.com/FreeOpcUa/opcua-asyncio (02.08.2023)

[3] https://flask.palletsprojects.com (02.08.2023)

proactive AAS architectures in [2]. "*The interaction manager uses state machines to implement various semantic interaction protocols for calling up the necessary decision algorithms*" [17]. Note that state machines can interact within an I4.0 component in the same way as with external interactions, using interaction protocols.



AASConfigureParser()
It extracts and parses the AASX file.

StateMachine()
It initializes and maintains all available state machines and their encapsulated capabilities and skills.

DataAdaptor()
It creates and maintains the data structures required for storing the AASX package data

DataManager()
It synchronizes the read and write methods of the internal data structures. The Python AAS server maintains internal data structures for representing AASX package data. The Data Manager acts as an interface layer for this internal data structures.

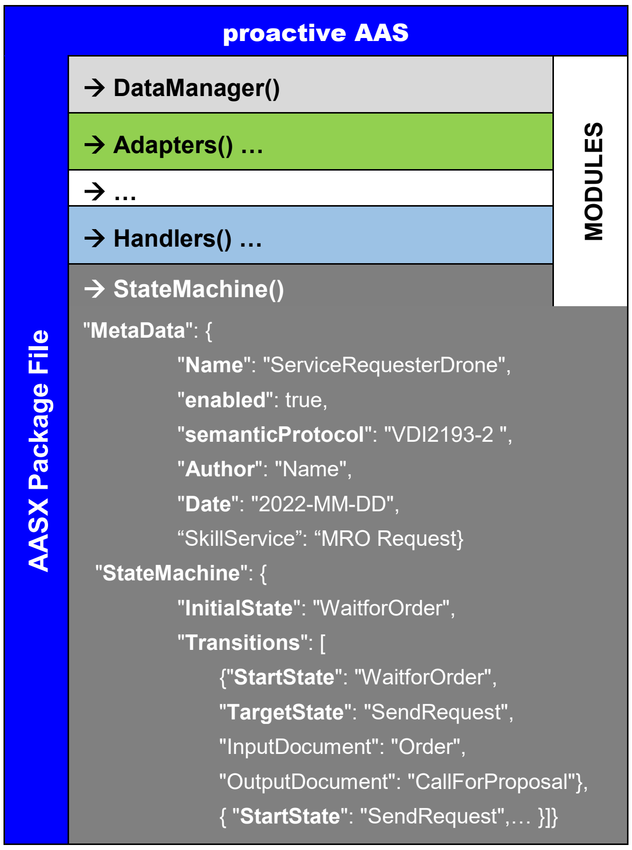
MsgHandler()
It manages the communication between all the modules and is the internal message bus of the server. It maintains two queues (inbound/outbound) and event listeners are attached to both of them (Event: receiving, sending message). The EndPoint Handlers need to "getIbMessage" method to push an inbound I4.0 messages into the message bus. The eventHandler "sendOutBoundMessage" invokes the dispatch method from the appropriate endpointhandler to send the outbound I4.0 messages.

AASEndPoints()
MQTT and HTTP endpoints of the Server.

AssetAccessPoints()
Configures the Asset Access Adaptors

Figure 5: Scheme of the Python-based proactive AAS, its interacting modules, AASX package file and connected to the associated asset via asset data server

Generally speaking, state machines are used in the Python AAS to functionalize or, if already present, encapsulate and top manage the capabilities and skills of the asset, or even just its digital representation (see chapter 2.2.1). A state machine consists of an initial state, a set of states and a set of transitions that enable movement from one state to another state. For each state machine, a Python script is implemented. This script is created based on a JSON representation (Figure 6 below): The JSON object has two parts, the *MetaData* and the actual state machine information containing a set of transitions. Each transition has a start state, target state, *InputDocument* (in case the start state is expecting a message of a specific type for further processing) and an *OutputDocument* (In case the start state would like to send a message to another state machine).

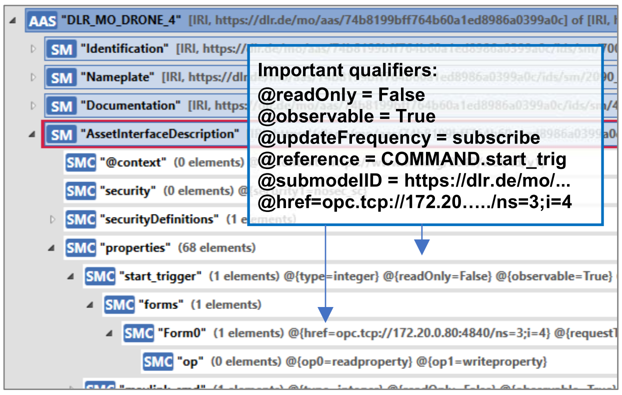


```
"MetaData": {
        "Name": "ServiceRequesterDrone",
        "enabled": true,
        "semanticProtocol": "VDI2193-2 ",
        "Author": "Name",
        "Date": "2022-MM-DD",
        "SkillService": "MRO Request}
 "StateMachine": {
        "InitialState": "WaitforOrder",
        "Transitions": [
            {"StartState": "WaitforOrder",
             "TargetState": "SendRequest",
             "InputDocument": "Order",
             "OutputDocument": "CallForProposal"},
            { "StartState": "SendRequest",… }]}
```

Figure 6: JSON representation of the state machine

The created Python script consists of a set of classes, one for each state, and a base class that is responsible for starting the state machine and iterating over the states using transitions. Each class related to the state has two main functions **run**() and **next**(). The run() method is setup to start the execution of the current state and the next() method sets the state to its next states. A separate function called **logic**() is specially added to each of the class, a developer is expected to add all the extra logic pertaining to the state in this method. In the logic() function, for example, parameters of the asset are monitored and the next state is initiated or a (remote) sub-process is invoked when a threshold is reached. The transition from the current to the next state could be one out of many states, for this a Boolean variable is associated to each such state. Which next state is activated depends on the returned result of the current state, e.g. "rejected" = true or false. If inbound or outbound messages are required, which is defined based on the information provided in the json representation, they are pushed to or received from the respective message queues by the message handler.

The state machines, like all other modules of the Python AAS, use static and dynamic properties of the asset. To physically connect the AAS to the runtime data and functions of the asset, adaptors are implemented that are characterized by a meta-description in terms of another SM called **AssetInterfaceDescription**. This submodel is based on the W3C Web of Things standard (https://www.w3.org/WoT) and was derived by mapping both meta-models as presented in [30]. As a result, a *SubmodelElementCollection* (SMC) must be created for each element of the asset to be accessed from the proactive AAS (Figure 7, e.g. "start_trigger"): The SMC contains all information required to specify the asset element and associated access point, such as the sub-SMC *Form0* with e.g. an OPC UA endpoint and the related node ID or a REST API URL. Additional qualifiers determine the update frequency of the value within the AAS (interval, subscription), its accessibility as well as its reference to the property as used in the digital representation (AASX description). The AAS server updates the value of the referenced property whenever a new value is available from the access point.



```
AAS "DLR_MO_DRONE_4" [IRI, https://dlr.de/mo/aas/74b8199bff764b60a1ed8986a0399a0c] of [IRI,
  SM "Identification" [IRI, https://
  SM "Nameplate" [IRI, https://dlr
  SM "Documentation" [IRI, https:
  SM "AssetInterfaceDescription"
    SMC "@context" (0 elements)
    SMC "security" (0 elements) @
    SMC "securityDefinitions" (1 e
    SMC "properties" (68 elements)
      SMC "start_trigger" (1 elements) @{type=integer} @{readOnly=False} @{observable=True}
        SMC "forms" (1 elements)
          SMC "Form0" (1 elements) @{href=opc.tcp://172.20.0.80:4840/ns=3;i=4} @{request
            SMC "op" (0 elements) @{op0=readproperty} @{op1=writeproperty}
```

Important qualifiers:
@readOnly = False
@observable = True
@updateFrequency = subscribe
@reference = COMMAND.start_trig
@submodelID = https://dlr.de/mo/...
@href=opc.tcp://172.20…../ns=3;i=4

Figure 7: Submodel AssetInterfaceDescription in AASX

We establish communication in the I4.0 language between the AAS via MQTT. The Python AAS server itself does not host an MQTT server, but uses an external MQTT broker. The corresponding adapter subscribes to the identifiers/IDs of all AAS specified in the AASX file via the MQTT server. In an advanced more dynamic approach, the subscription depends on the required skills and capabilities of the participants, so that the identifiers/IDs are received from a register system.

2.1.2. Hardware development

The AAS can be stored and accessed in a variety of ways, depending on the specific implementation and requirements of the CPSS. Typically, the AAS is stored in a digital format within a centralized or distributed information system. This can be a cloud-based platform, a dedicated server, even on a PLC, or a combination of local and cloud storage. Here we present a very compact mobile system solution that runs the server described above and can be located right next to the asset. The idea is to simplify the digital upgrade of assets across domains to make them I4.0 ready - with an **off-the-shelf proAAS BOX** that only needs some customization.

At the heart of the server system is a ***Raspberry Pi 4B*** (RasPi), a single-board computer known for its reliability and flexibility. The RasPi serves as a central computing platform that has the features required to run the proactive Python AAS as an independent hardware platform.

For maximum flexibility, a ***SIM7600E-H 4G HAT*** LTE module is installed on the Raspberry to enable communication over mobile networks such as 5G. It has been assigned a **fixed IP** to be uniquely identifiable and, if granted, globally accessible via the Internet. LTE also ensures greater reliability and uninterrupted connectivity if another interface (e.g. LAN, WLAN) fails. In addition, the programmable power management module ***StromPi 3*** is added. It compensates for voltage fluctuations and automatically handles different power sources (6V to 61V). The **BOX has several interfaces** to connect downstream to the asset and upstream to the CPSS:

USB ports enable easy connection to various external peripherals, allowing seamless expansions and interaction with a diverse range of devices. **GPIO** pins allow for digital input and output, making it possible to interface the Raspberry Pi with sensors, actuators, and other external devices. By utilizing GPIO libraries and programming, developers can create custom interactions and integrate the BOX into a wide range of assets. **Network adapters** act as gateways to establish connections with the corresponding asset (hard- or software) or other AAS and devices in the CPSS. Using the network adapters, the BOX can send and receive data packets to exchange information, commands and updates with the connected endpoint. It can use communication patterns such as HTTP, TCP/IP or custom protocols. Leveraging all interfaces ensures seamless integration between assets and the AAS, enabling real-time monitoring, proactive decision making and enhanced asset interoperation within an Industry 4.0 ecosystem.

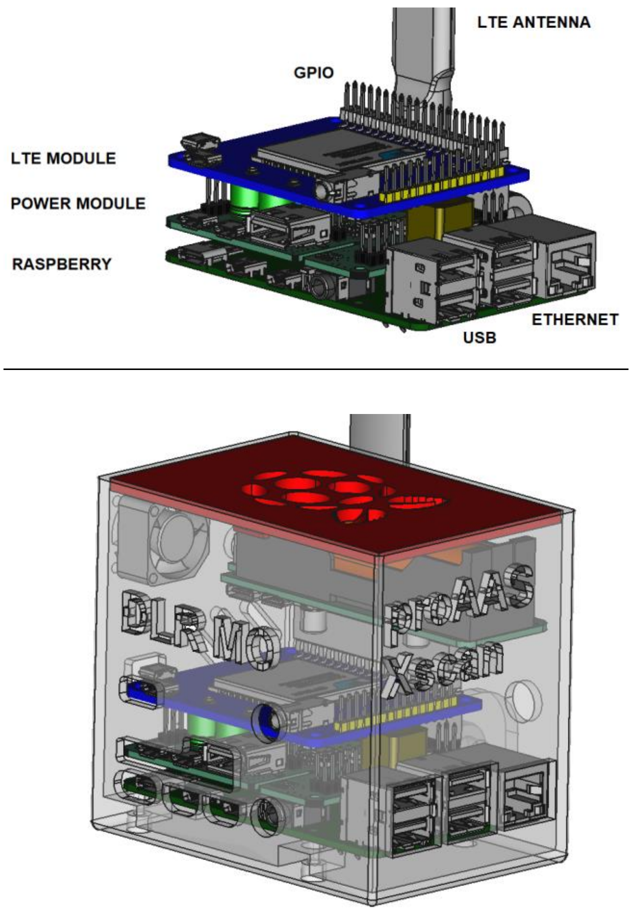


Figure 8: Developed proAAS BOX

2.2. Submodels

The metamodel of the Asset Administration Shell [18] consists of individual *Submodels* (SM) and their set of *SubmodelElements*, including property instances. This composition contains all the asset information to represent digitally its properties, capabilities, skills or general behavior. It should be noted that an asset can be broken down into other assets and the AAS associated with them. To achieve the overall AAS objective of standardized interoperability, submodel templates provide common specifications for guidance. These templates are officially elaborated, reviewed and published by the IDTA. The AAS distinguishes between type and instance in terms of an original type specification, e.g. from the manufacturer, and the adapted application as an instance of the specification in an operational environment – considering asset's lifecycle. All elements can be uniquely identified. They can have specific access controls and be maintained throughout their lifecycle. Even aspects such as procedures for handling over assets to another owner are defined, e.g. in the *Handover Documentation* [20]. In the presented paper, we will focus on submodels covering capabilities and skills in proactive AAS, including the exchange of digital certificates e.g. the EASA FORM 1.

2.2.1. Bidding State Machine Submodel

We build on the I4.0 scenario of independent, vertically and horizontally interacting partners in a CPSS. This also includes less complex interactions in scenarios where only information is exchanged. What they all have in common is the use of the I4.0 language. To enable the proactive AAS to autonomously interoperate with other proactive AAS, a state machine based on the scheme introduced in chapter 2.1.1 was implemented that can initiate, participate, control and direct bidding interactions. It is about finding the appropriate service provider to a service request in terms of technical requirements, costs or suggested alternatives. Requests can range from detailed to general (e.g. "drilling" or "MRO service"). The requester initiates the bidding process by sending a call for proposal (CfP) to the CPSS. The receiving partners internally check their capabilities and skills to evaluate if they can fulfill the request and return a proposal for further negotiations until the final decision is made. Thus, decision making is crucial at certain points and requires appropriate techniques of Multi-Criteria Decision Analysis (MCDA), such as TOPSIS, that are applicable in IT algorithms. As described later, the negotiation skill of the bidding capability could be thought of as one such central skill, which precedes all other skills. However, a pre-defined granularity of the bidding process allows flexibility in whether a skill can be requested directly or needs to be negotiated. A submodel for the implementation of a bidding state machine in proactive AAS is proposed below. It is based on the IDTA's published "Control Component" template [21].

The VDI/VDE 2193:2020 part 2 [35] formalizes the bidding procedure between I4.0 components (Figure 9 a): According to this, the flow of interaction is not linear, but involves feedback loops of negotiation, clarification, withdrawal, revision or rejection. Based on this framework and considering decision nodes, the steps of the interaction protocol can be specified as follows:

1. **Call for Proposal**: The Service Requester (SR) sends a call for proposal (CfP) to potential Service Providers (SP). The CfP document outlines the desired services, requirements, and other relevant information.
2. **Proposal Generation**: The SP receives the CfP and assesses whether they understand the requirements and if they can deliver the services based on their capabilities, skills and resources. Based on this evaluation, the SP creates a proposal that includes the services, costs, timeframes, and other contract relevant details.
3. **Evaluation with MCDA Techniques**: The SR receives multiple proposals from potential Service Providers. Using MCDA techniques, the SR evaluates the proposals based on predefined criteria such as cost, quality, provider experience, delivery time, and other factors.
4. **Contract Negotiation**: The SR selects the best proposal based on the evaluation using MCDA techniques. The SP with the most favorable proposal receives a contract that is sent to them. The SP can accept the contract, reject it, or propose revisions to align with their specific needs.
5. **Contract Finalization**: If the SP accepts the contract or agrees to the revisions, the contract is finalized between the SR and the SP.
6. **Service Execution**: The SP commences the execution of the contract as per the agreed-upon terms. They deliver the services within the specified timeframe and in adherence to quality standards.
7. **Result Verification**: Upon completion of the service execution, the SR verifies the results (e.g. quality check). They may accept the delivery, reject it, or request rework if the results do not meet the expectations.

In order to operationalize these steps in a bidding state machine of the Python AAS, a set of classes is created as introduced in section 2.1.1, one for each interaction step. This is shown for the service provider (contractor) in Figure 9 (b). Whit respect to the service requester state machine, some states are added, e.g. the *WaitForNewOrder* or *CallForProposal*, others are removed, e.g. the *ServiceProvision*.

The state machines of the SR and SP have in common that they exchange their negotiation relevant documents. These documents are part of or have their own referenced submodel, whose JSON serialization is part of the I4.0 language pattern (I4.0 message elements). A state for **capability check** evaluates these documents e.g. for readability and completeness. A state for **feasibility check** assesses whether the skills available are suitable to execute the request. A state for **schedule check** evaluates the date, time, duration and location of the requested provision. With this information and process-specific requirements, the expected **cost indicators** are calculated, the proposal is prepared and returned to the SR in another states (Figure 9 (b) middle). The SR evaluates the returned documents in the same way, but from a different perspective. Note that there may be more than one proposal returned if more than one SP responds. If there is no exact match between the required and proposed features, both the SR and the SP can decide whether to reject, revise or accept the offer after considering their relevant criteria by using IT-based **MCDM techniques**. MCDM ranks the proposals received on the basis of the performance indicators provided, considering the service requester's specific weighting of these indicators. For example, if the weight of the price is lower than the delivery time, proposals with even higher prices but faster delivery could be selected. MCDM techniques such as TOPSIS can process a much higher number of criteria and

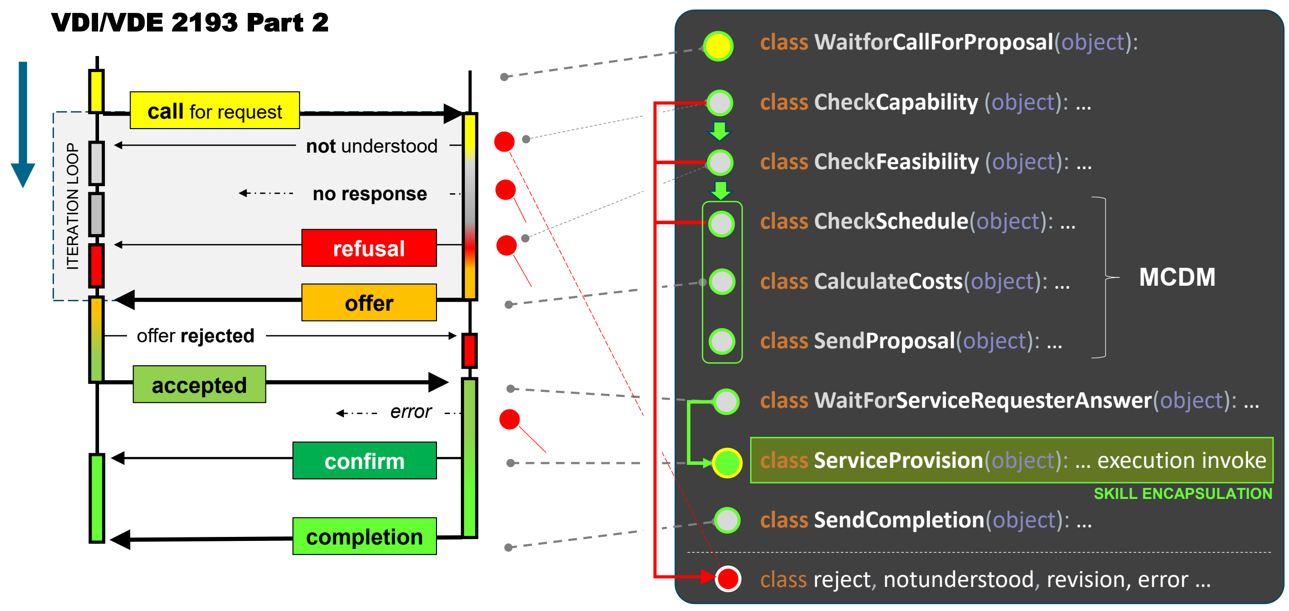

VDI/VDE 2193 Part 2
ITERATION LOOP
call for request
not understood
no response
refusal
offer
offer rejected
accepted
error
confirm
completion
class WaitforCallForProposal(object):
class CheckCapability (object): ...
class CheckFeasibility (object): ...
class CheckSchedule(object): ...
class CalculateCosts(object): ...
class SendProposal(object): ...
MCDM
class WaitForServiceRequesterAnswer(object): ...
class ServiceProvision(object): ... execution invoke
SKILL ENCAPSULATION
class SendCompletion(object): ...
class reject, notunderstood, revision, error ...
(a)
(b)

Figure 9: (a) Interaction protocol based on VDI/VDE 2193 [35], (b) mapped to classes as states of a SP state machine

ultimately provide a ranked list of alternative proposals on the basis of which a decision is made. If not accepted in the first round of bidding, either a revision is iteratively worked out between SR and SP until a common agreement is reached or a rejection is decided. In the case of an agreement, the next state is **service provision**. It encapsulates the actual required, executable skill in terms of triggering and monitoring its associated functions. All requests, even switching a skill on and off, require a negotiation (simple or complex). Since negotiation can also be interpreted as a skill, it can be assumed that it is always the primary involving the secondary skill. (Figure 9 (b) → *skill encapsulation*).

The "wrapped" skills can be invoked in different ways: Calling other state machines, direct connections (OPC UA, REST, ...) or skill-specific submodels. A common description of capabilities and skills is proposed for their uniform handling, referring to a reference model for common understanding in [27]. Along with that, a comprehensive ontology for modelling capabilities and skills in cyber-physical systems is introduced in [28]. On the other hand, the IDTA published the **Capability** and **Control Component** submodels (CCS) as standardized representations of capabilities and skills in the AAS. In the CCS, skills are collected in the SMC Skills and a single skill is modelled via the SMC Skill. To reach the highest outcome, we compared in [37] the ontology model to the IDTA submodel and identified strong commonalities but also missing elements. The further modelling of capabilities and skills for the AAS will build on these experiences, starting we the development of a submodel of the bidding capability and their corresponding skills.

The IDTA CC submodel is used here to propose a normative and interchangeable structure for the implementation of the skills of the bidding capability in terms of the state machine described above. For its creation in accordance with this submodel, the most relevant properties are shown in the UML-diagram in Figure 10: The SMC *Skills* comprise all the skills provided, each defined in a subordinated SMC *Skill*. The latter contains the basic information needed to call the skill, such as configuration parameters, error codes, or collections of references to other skills used by that skill. For its transfer to a Bidding Model, two skills are implemented: One to lead the negotiation from the customer's perspective (requester) and the other to lead the negotiation from the contractor's perspective (provider). The current role of the asset is indicated by the *Disabled* property, which is set to either true or false. The SMC *Parameters* comprises a list of parameters that are used here to configure the states of the bidding state machines. Each parameter is defined by a name, type, direction and values. Here the *Type* is a state, the *Name* is assumed to be the name of the state and the *Direction* is usually *InOut* - indicating that the state is a receiver and provider of data. The associated SMC *Values* consists of all the meta-info for initializing the states and its transitions. However, unlike the original IDTA CC template, we have subordinated the SMC *Values* with additional SMCs *Parameters* instead of *Properties*. This allows to describe the states' transition more precisely in terms of a state-transition table (Figure 12). These parameters are defined with the properties *Type*, *Name*, *Direction* and *Enabled*.

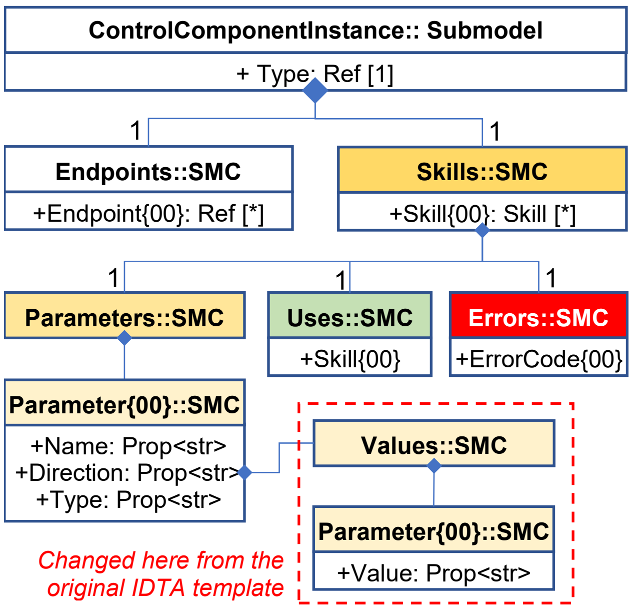
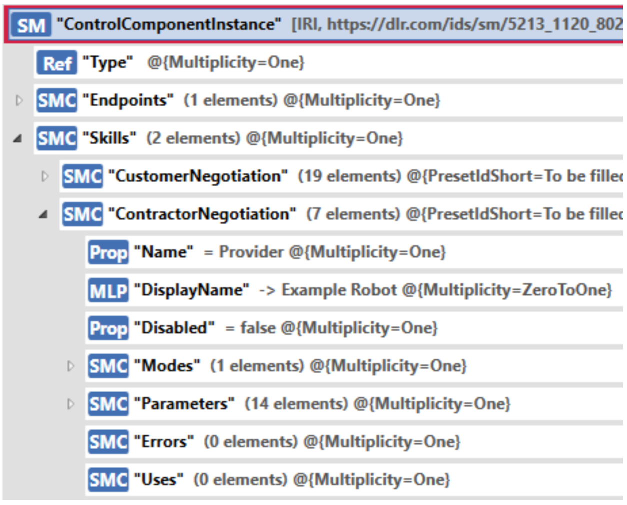


Figure 10: UML-Diagram of adapted IDTA submodel *ControlComponentInstance*, based on [22]

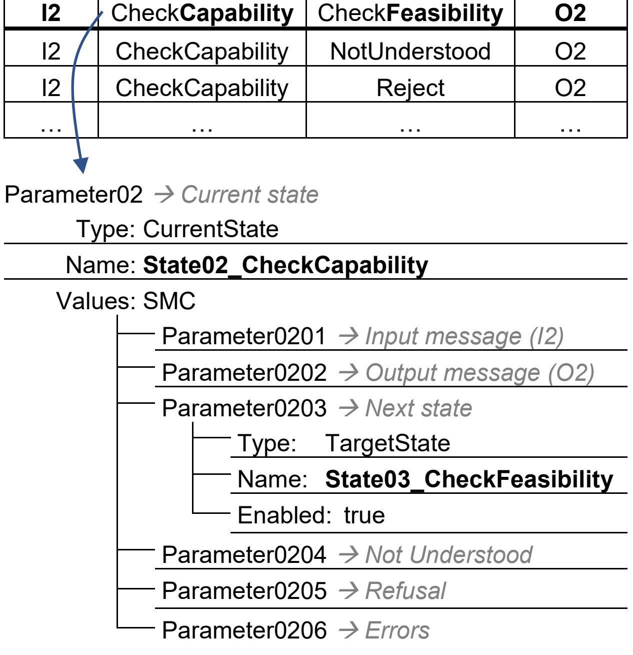


SM "ControlComponentInstance" [IRI, https://dlr.com/ids/sm/5213_1120_802
Ref "Type" @{Multiplicity=One}
SMC "Endpoints" (1 elements) @{Multiplicity=One}
SMC "Skills" (2 elements) @{Multiplicity=One}
SMC "CustomerNegotiation" (19 elements) @{PresetIdShort=To be fille
SMC "ContractorNegotiation" (7 elements) @{PresetIdShort=To be fille
Prop "Name" = Provider @{Multiplicity=One}
MLP "DisplayName" -> Example Robot @{Multiplicity=ZeroToOne}
Prop "Disabled" = false @{Multiplicity=One}
SMC "Modes" (1 elements) @{Multiplicity=One}
SMC "Parameters" (14 elements) @{Multiplicity=One}
SMC "Errors" (0 elements) @{Multiplicity=One}
SMC "Uses" (0 elements) @{Multiplicity=One}

Figure 11: Skills definition (Customer/Contractor) of the bidding capability, based on the IDTA submodel *Control-ComponentInstance* [22]

- *Type* identifies the parameter such as *Current state*, *Target state* or *I4.0 message*.
- *Name* details the associated type, e.g. *Input* or *Output* of the type *I4.0 message* or *Capability Check* of the type *Current* or *Target state.*
- *Direction* indicates whether the parameter is a receiver or a provider, or both, in the data stream.
- *Enabled* indicates whether the parameter is active or not. For example, if the parameter of type *Target state* with the name *CheckFeasibility* is true, it will be the next.

Input	Current state	Next state	Output
I1	WaitForCfP	CheckCapability	O1
I2	Check**Capability**	Check**Feasibility**	**O2**
I2	CheckCapability	NotUnderstood	O2
I2	CheckCapability	Reject	O2
…	…	…	…



Parameter02 → *Current state*
Type: CurrentState
Name: **State02_CheckCapability**
Values: SMC
Parameter0201 → *Input message (I2)*
Parameter0202 → *Output message (O2)*
Parameter0203 → *Next state*
Type: TargetState
Name: **State03_CheckFeasibility**
Enabled: true
Parameter0204 → *Not Understood*
Parameter0205 → *Refusal*
Parameter0206 → *Errors*

Figure 12: State Transition Table and an example of how to transfer to the adapted IDTA CC submodel ([21], [22])

As presented above, the capabilities and skills originally requested by the SR from the SP are encapsulated in the bidding process, where the execution of the associated skill(s) is triggered in the state class *ServiceProvision*. These skills can be defined by further CC submodels, as introduced earlier, containing not only the metainformation about the skill itself used to negotiate in different states, but also the information about how to invoke it. To make them available to the bidding state machine, the SMC *Uses* of the CC submodel IDTA is intended to collect all the skills in terms of their references (e.g. submodel ID).

2.2.2. Tender Document Submodel

For the bidding process, the service requester and service provider need a document that lists all the properties to be negotiated. This document must have a predefined structure that is known to all participants. The document is represented here by a submodel template. Its proposed content is generalised to be independent of the negotiation topic. The latter has to be customized by the user. The submodel data is exchanged as the message part of the I4.0 language pattern.

In general, a service is negotiated and therefore its description needs to be interpreted in the same way across business partners, technologies or lifecycle phases within the interoperation [10]. Building on the IEC 61360 specification, [3] recommends the description of services with a property model. In line with this, Figure 13 shows the description of a service as a combination of meta, process and functional properties, or, in the words of the IEC 61360, attributes. Thus, the properties' description goes beyond the "*purely technical capabilities and may include, for example, economic criteria such as delivery dates, cost and agreements regarding documentation or maintenance*" [8].

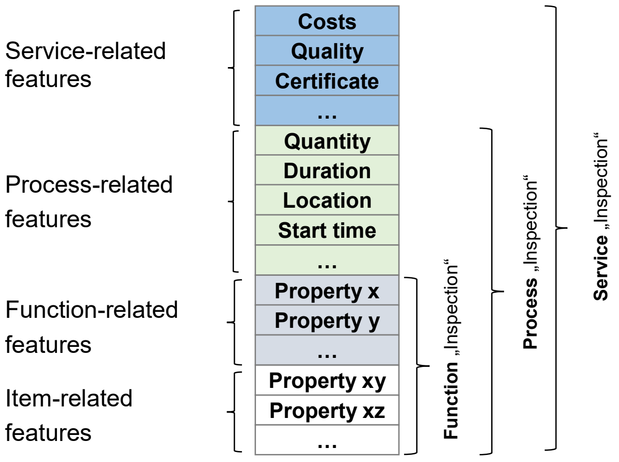


Figure 13: Example of modelling a service description, based on [3], [33]

The description can be implemented into the Asset Administration Shell by two Submodels, one considering the perspective of the SR, the other that of the SP. In fact, the submodel elements of both should be the same, except for the range of values of the properties themselves. For example, the SR specifies a single property value and the SP can supply the property in a certain range. Conversely, the SP offers a property with a single value, while the SR accepts a range of values for that property. If within the range, the decision-making procedures select the appropriate values, otherwise reject the SR's request or SP's offer.

Following [27], the different properties of a service can be categorized by Submodel Collections (SMC), such as a

SMC *TenderCriteria*, which includes, for example, required quantities, price specifications, environmental limits or delivery time or locations. In a further step, e.g. [29] and [33] divide the service description into a technical and a commercial part, which can be seen as two SMCs: The commercial part describes the conditions of the service provision and specifies the desired delivery time, a place and a price of a service provision. In a broader context, environmental or social properties could also be considered. The technical part includes information about the related item itself (e.g. type, dimensions) as well as functional properties that quantify the requested measures on the item in as much detail as necessary. Note that the item can be understood as a workpiece in a manufacturing scenario and as an asset to be maintained in a maintenance scenario. Thus, *Item* is a more neutral expression in a general approach.

Based on the above findings, a submodel template has been created to serve as a standard tender document. Figure 14 shows the general concept with some exemplary properties: The submodel labelled with the name of the service is first divided into the *TechnicalProperties* and *CommercialProperties* SMCs. Other SMCs may be added if necessary. The SMC *TechnicalProperties* is initially clustered with the SMC *ItemProperties* and *FunctionalProperties*. The template must be customized for all available services, for both the SR and the SP. For example, Figure 14 shows some properties from the perspective of a SP that offers a range of values defined by lists or upper and lower limits. Chapter 3 shows how this template can be adapted and applied to autonomous bidding procedures between an aircraft system surrogate and other participants in the MRO CPSS.

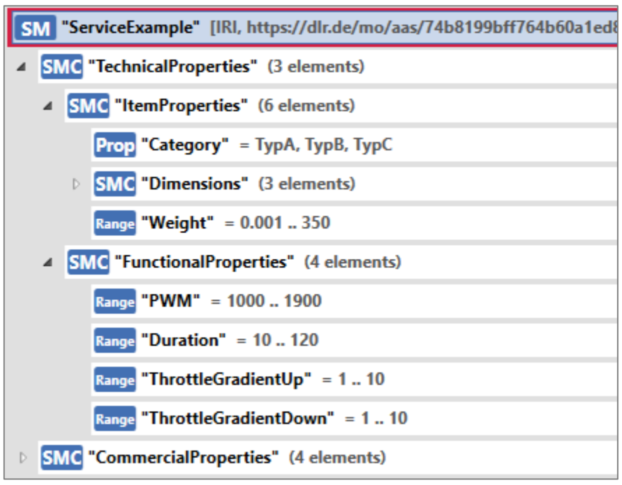

```
SM "ServiceExample" [IRI, https://dlr.de/mo/aas/74b8199bff764b60a1ed8
  SMC "TechnicalProperties" (3 elements)
    SMC "ItemProperties" (6 elements)
      Prop "Category" = TypA, TypB, TypC
      SMC "Dimensions" (3 elements)
      Range "Weight" = 0.001 .. 350
    SMC "FunctionalProperties" (4 elements)
      Range "PWM" = 1000 .. 1900
      Range "Duration" = 10 .. 120
      Range "ThrottleGradientUp" = 1 .. 10
      Range "ThrottleGradientDown" = 1 .. 10
  SMC "CommercialProperties" (4 elements)
```

Figure 14: SM Tender Document; example for SP

3. EXPERIMENTAL IMPLEMENTATION

Chapter 2 presents a theoretical view of how I4.0 components could be used in an MRO environment to answer the research questions at the beginning. As a practical proof of concept, the interaction between different partners in a data-driven MRO CPSS is demonstrated through the implementation in our maintenance simulation model *MaSiMO*, introduced in [38]: Here an aircraft surrogate, a robotic system and an HMI are equipped with the previously introduced proactive AAS BOX. In this way, the participants are advanced to I4.0 components. In the scenario, the aircraft surrogate detects a problem and its monitoring proactive AAS requests a part replacement (CfP). The only recipient of this CfP is the robotic system which, after internal checks, returns a service proposal to the service requester (aircraft surrogate). The SR accepts the proposal and responds with an execution order. During the service provision, the robot requests assistance from a human to complete its task.

3.1. I4.0 Components in the CPSS

3.1.1. Aircraft system as MRO service requester

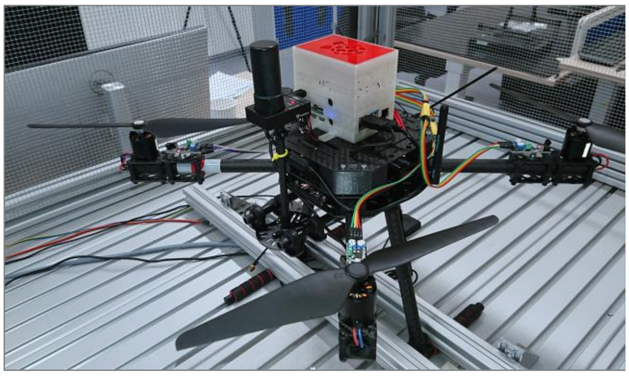


Figure 15: Aircraft surrogate system with proAAS BOX

An unmanned aerial vehicle, namely the Holybro X500, has been upgraded to an I4.0 component by installing a proAAS BOX (Figure 15). In the use case, it plays the role of both MRO service requester and, to some extent, service provider. The asset is a complex surrogate of an aircraft system with a plenty of sensors, flight data and control functions. Additional sensors were integrated to overcome missing data, such as directly measured engine rotation speed. The sensors provide data on acceleration, vibration, environmental conditions, power status and more. The control functions execute external and internal commands, from simply turning a motor at a desired speed to a fully programmed flight mission. Everything is handled by an aircraft controller, namely a Pixhawk, which is connected to the proAAS BOX via its serial interface (USB).

The data server introduced in 2.1 is used to extend the UAV's ability to communicate bidirectionally via OPC UA and REST interfaces. Both are realized in Python (asyncua, flask) and use the dronekit library[4] to access the UAV's controller. These interfaces are used to connect the UAV with the proactive AAS. The OPC UA server structures the properties into three domains as object nodes:

- Command
- Operational Data
- Parameter

The *Command* node groups all the variables that actively control the asset, from controlling the onboard data recorder to the UAV movements. The *Operational Data* node consists of monitoring and telemetry properties. And the *Parameter* node contains all the settings, such as calibration factors, to tune the behaviour of the UAV.

For management and decision making, the proAAS BOX continuously requires the status and health of the aircraft system surrogate, which is provided by appropriate OPC UA values: An *Operational Data* node is divided into the groups *Monitoring* and *Telemetry*. These groups further cluster the data to specific system components or system

[4] Version 2.9.2; https://pypi.org/project/dronekit/ (02.08.2023)

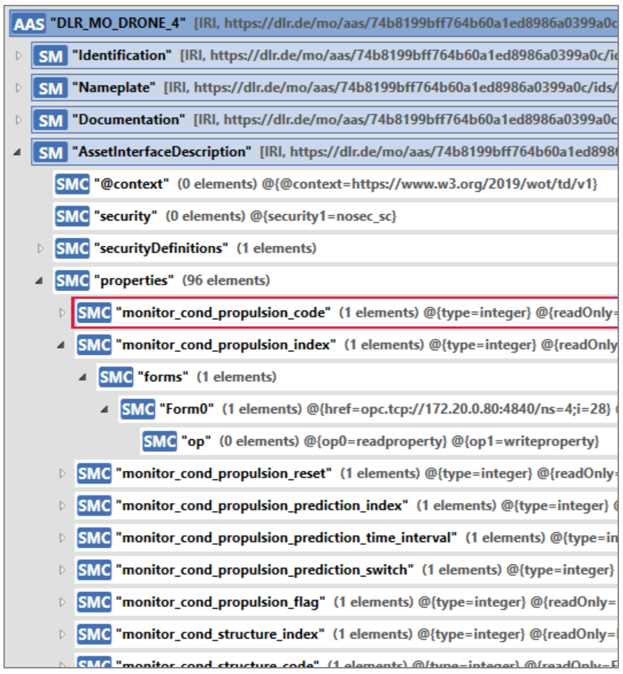
properties such as *Propulsion*, *Structure*, *Power supply* or *System time*, *Attitude* or *Ambient pressure*. The *Monitoring* node distinguishes between *Condition indicators* and *Efficiency indicators*. The condition indicators are aggregated system indices that are calculated by separate running algorithms to indicate the current or predicted health conditions of the associated aircraft system. The framework of the health algorithms was developed at the DLR Institute MO [26] and adapted specifically for implementation in the proAAS BOX Data Server. In the current work, this framework uses sensor data from the power supply system. In addition, a simplified fault detection system based on learning algorithms is implemented and trained to raise a fault flag in the event of abnormal vibrations. The efficiency indicators go in a similar direction, but only directly record the physical efficiencies such as the ratio of thrust to required power.

The proactive AAS independently decides if and when an MRO visit is required and thus an order must be triggered. For this purpose, a monitoring state machine runs continuously in the AAS, evaluating all of the above OPC UA variables of the condition and efficiency metrics. When an event occurs, it sends an order (e.g. a propeller inspection) to the bidding state machine as it is introduced in chapter 2.2.1: The state *WaitForNewOrder* receives the order, processes it and forwards it as a request to the next state *CallForProposal*. Before this all can happen, the proAAS must be configured and initialized with the digital representation ("digital twin") of the UAV defined in the AASX format. Besides its general description such as unique identification, nameplate or documents like CAD files, manuals or certificates like the EASA Form1, the AASX contains the submodels of interface descriptions and the capabilities and skills. Here, we focus on the latter only.

The physical link between the aircraft system surrogate and the proAAS server is configured in the AASX by the *AssetInterfaceDescription* submodel (see chapter 2.1), which consists of 96 OPC UA variables. As partially shown in Figure 16 for the propulsion condition indicators, all 96 variables are defined with their associated endpoint (OPC UA server URL + node ID) and features (e.g. writable = true). In addition, the *AssetProperties* submodel defines all the variables that can be used throughout the proAAS server framework, e.g. in the implemented state machines. These variables are connected to the corresponding values of the asset provided by the appropriate interface adapter (e.g. OPC UA) configured in the SM *AssetInterfaceDescription*. Here we have structured the internal values in the same way as the OPC UA data server.

For the use case, a simplified logic is built into the proAAS to monitor selected system health indicators, processed by the algorithm mentioned above, and make decisions based on their values. For simplicity, the logic is part of the *WaitForNewOrder* state of the embedded bidding state machine. When limit values are reached or are projected to be approached (Figure 17), the proAAS of the aircraft surrogate decides what action to take, such as requesting MRO treatments. The aircraft sends then a CfP with its requirements to the CPSS network via a dedicated MQTT broker to which several potential MRO providers are actually subscribed (in the use case only one).

The requirements are specified in an AAS submodel, namely *RequesterMRObasic*, in accordance with the introduced tender document in Chapter 0. As shown in Figure 18, it comprises the two SMC *TechnicalProperties* and *CommercialProperties*. The *ItemProperties*, a SMC of the *TechnicalProperties*, contains the meta-description of the aircraft, such as its certified type, dimensions or weight properties. It also includes the classification of the systems on which the MRO provider is required to have the capability, skills and authorisations to work. In aviation, these system classes can be associated with the FAA JASC/ATA 100 reference system [15], which groups the aircraft elements into chapters. For example, Figure 18 shows that the service requester is a UAV, is certified to CS-VLR, weighs 1,5 kg and requires treatments for systems in terms of ATA Chapters 28 (fuel system), 53 (fuselage), 64 (tail rotor) and 80 (starter system). In the SMC *FunctionalProperties*, the service requester details the required specifications of the requested skill, such as the expected type or scope of inspection, or for a repair process, the requirements for physical treatments or tolerances.



```
AAS "DLR_MO_DRONE_4" [IRI, https://dlr.de/mo/aas/74b8199bff764b60a1ed8986a0399a0c
  SM "Identification" [IRI, https://dlr.de/mo/aas/74b8199bff764b60a1ed8986a0399a0c/ic
  SM "Nameplate" [IRI, https://dlr.de/mo/aas/74b8199bff764b60a1ed8986a0399a0c/ids/
  SM "Documentation" [IRI, https://dlr.de/mo/aas/74b8199bff764b60a1ed8986a0399a0c
  SM "AssetInterfaceDescription" [IRI, https://dlr.de/mo/aas/74b8199bff764b60a1ed898
    SMC "@context" (0 elements) @{@context=https://www.w3.org/2019/wot/td/v1}
    SMC "security" (0 elements) @{security1=nosec_sc}
    SMC "securityDefinitions" (1 elements)
    SMC "properties" (96 elements)
      SMC "monitor_cond_propulsion_code" (1 elements) @{type=integer} @{readOnly=
      SMC "monitor_cond_propulsion_index" (1 elements) @{type=integer} @{readOnly
        SMC "forms" (1 elements)
          SMC "Form0" (1 elements) @{href=opc.tcp://172.20.0.80:4840/ns=4;i=28}
            SMC "op" (0 elements) @{op0=readproperty} @{op1=writeproperty}
      SMC "monitor_cond_propulsion_reset" (1 elements) @{type=integer} @{readOnly=
      SMC "monitor_cond_propulsion_prediction_index" (1 elements) @{type=integer}
      SMC "monitor_cond_propulsion_prediction_time_interval" (1 elements) @{type=in
      SMC "monitor_cond_propulsion_prediction_switch" (1 elements) @{type=integer}
      SMC "monitor_cond_propulsion_flag" (1 elements) @{type=integer} @{readOnly=
      SMC "monitor_cond_structure_index" (1 elements) @{type=integer} @{readOnly=
      SMC "monitor_cond_structure_code" (1 elements) @{type=integer} @{readOnly=F
```

Figure 16: Part of the UAV AASX file

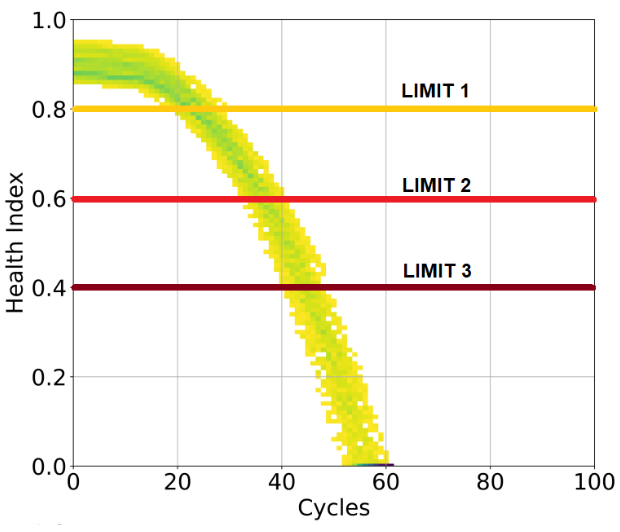


Figure 17: Example of system health development [26]

For the *CommercialProperties*, the tender document will specify preferred delivery locations, countries or a range of expected delivery time and prices. For target setting, the aircraft could also request optimised values from external cost-benefit services such as LCC or LCA, if available in the CPSS and accessible via AAS. Note that this is only a reduced number of possible properties for this example. They can also be extended to include properties with environmental or social impacts in order to consider sustainable solutions.

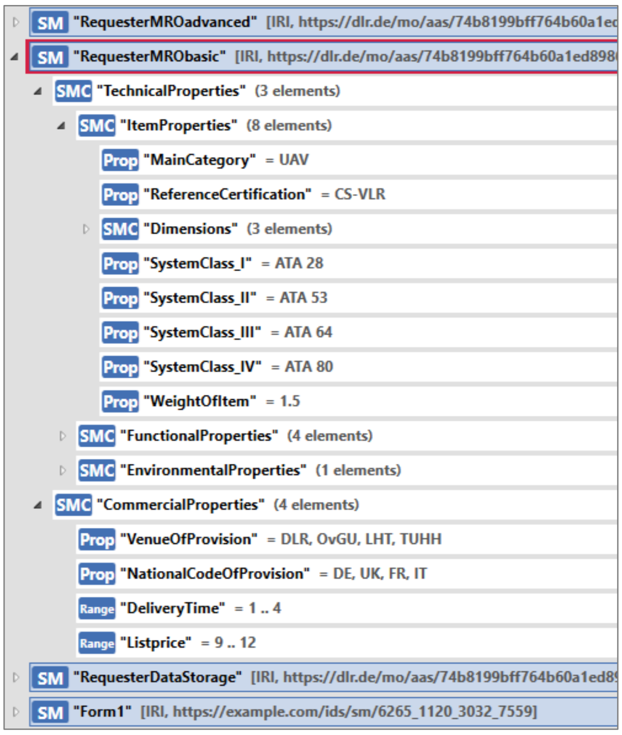

SM "RequesterMROadvanced" [IRI, https://dlr.de/mo/aas/74b8199bff764b60a1ec
SM "RequesterMRObasic" [IRI, https://dlr.de/mo/aas/74b8199bff764b60a1ed898
SMC "TechnicalProperties" (3 elements)
SMC "ItemProperties" (8 elements)
Prop "MainCategory" = UAV
Prop "ReferenceCertification" = CS-VLR
SMC "Dimensions" (3 elements)
Prop "SystemClass_I" = ATA 28
Prop "SystemClass_II" = ATA 53
Prop "SystemClass_III" = ATA 64
Prop "SystemClass_IV" = ATA 80
Prop "WeightOfItem" = 1.5
SMC "FunctionalProperties" (4 elements)
SMC "EnvironmentalProperties" (1 elements)
SMC "CommercialProperties" (4 elements)
Prop "VenueOfProvision" = DLR, OvGU, LHT, TUHH
Prop "NationalCodeOfProvision" = DE, UK, FR, IT
Range "DeliveryTime" = 1 .. 4
Range "Listprice" = 9 .. 12
SM "RequesterDataStorage" [IRI, https://dlr.de/mo/aas/74b8199bff764b60a1ed89
SM "Form1" [IRI, https://example.com/ids/sm/6265_1120_3032_7559]

Figure 18: Tender document submodel for a basic MRO request with aircraft as service requester (example)

Conversely, the aircraft can also act as a service provider, even as part of an MRO service. In this situation, the MRO provider acts as a requester and requests specific aircraft executions, from the delivery of documents (e.g. CAD files, certificates) to the operation of systems (e.g. engine runs, test or alignment flights). We have implemented the first capabilities, such as *ServiceEngineGroundTest*, which can be requested by the MRO provider using the bidding process. As shown in Figure 19, the tender document is adapted to the associated skill in terms of the functional characteristics offered by the aircraft, such as the required RPM, the duration of the test run and the throttle gradients (acceleration, deceleration).

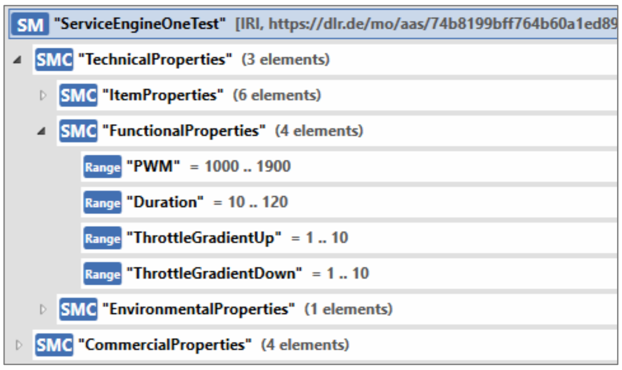

SM "ServiceEngineOneTest" [IRI, https://dlr.de/mo/aas/74b8199bff764b60a1ed89
SMC "TechnicalProperties" (3 elements)
SMC "ItemProperties" (6 elements)
SMC "FunctionalProperties" (4 elements)
Range "PWM" = 1000 .. 1900
Range "Duration" = 10 .. 120
Range "ThrottleGradientUp" = 1 .. 10
Range "ThrottleGradientDown" = 1 .. 10
SMC "EnvironmentalProperties" (1 elements)
SMC "CommercialProperties" (4 elements)

Figure 19: Tender document submodel for engine run-up request with MRO as service requester and aircraft as service provider (example)

These and other properties, such as monitoring values, are controlled in the related skill state machine via OPC UA. The entire flow of bidding interactions, including I4.0 compliant data exchange via MQTT, all checks, decision making or skill execution is the same as described in the previous chapters. Only the role of the aircraft has changed from SR to SP. During interoperation in the CPSS between all participants to perform MRO processes, the aircraft surrogate regularly changes its role to support the capabilities summarised in Table 1. Their implementation is either under development or planned, but all have in common that they are encapsulated by the skills of the bidding capability.

Aircraft role	**Capability**
By default	
SR+SP	Bidding Procedure
Extensions in progress	
SR	Request MRO basic
SR	Request external data storage
SP	Provision engine run-up
SP	Provision of document EASA FORM1
Planned extensions	
SR	*Request MRO advanced*
SR	*Request external decision support*
SP	*Provision of documents, data and files*
SP	*Provision of physical movements*
SP or SR	*t.b.d.*

Table 1: Roles of the aircraft surrogate in the CPSS and associated capability to request or provide services

3.1.2. Robot as MRO service provider

In the use case, a *Universal Robots UR10e*[5] plays the role of an automated MRO service provider. Its application focuses on enabling its autonomous interaction with other assets in the CPSS using a proactive AAS, rather than developing professional robot-based MRO skills. The latter are created as complex functions in separate projects, running on the robot's PLC or other performant controller. However, these skills will be implemented in the proactive AAS by encapsulating them as described in section 2.2.1. There, the skills are invoked by triggering the PLC or controller via appropriate interfaces (e.g. OPC UA, REST).

First, as with the aircraft system surrogate, a data server was built in Python. It uses a Python communication SDK developed and licensed by UnderAutomation[6]. It provides libraries to remotely control the robot and its effectors, with commands transmitted via Python dashboard controls or socket adapters. Additionally, a RDTE interface receives data streams from the robot to read real-time information on positions, status, inputs and outputs, the value of programme variables, temperatures, voltages, currents, etc. Based on these interfaces, the programmed OPC UA server provides all properties as node ids via a network accessible endpoint. These OPC UA values are made available in the proactive AAS of the robot by customizing the *AssetInterfaceDescription* and *AssetProperties* submodels in the AASX file according to section 2.1.1 (Figure 7). The 76

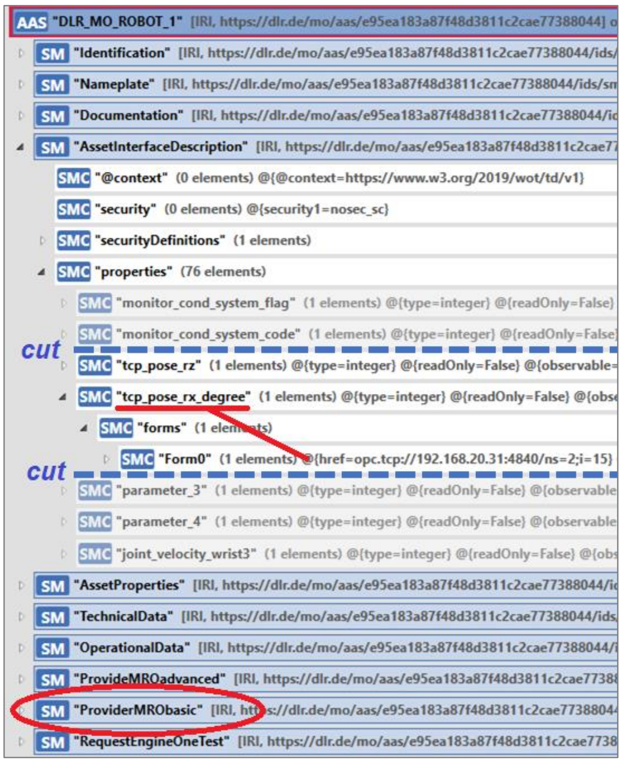
[5] https://www.universal-robots.com/products/ur10-robot/ (Access 30.07.2023)

[6] https://underautomation.com/ (Access 30.07.2023)

OPC UA variables consist of parameter, supervisory and control values used by the AAS to manage the UR10e as SP and SR in the CPSS. This includes autonomous decisions, bidding interactions in terms of service provision (e.g. inspection, repair) and service requests (e.g. human support) or data persistence management. As the AAS is a digital representation (digital twin) of the robot UR10e, general data such as ID, documents, certificates are also defined by standardized submodels. For example, the Internationalised Resource Identifier (IRI, RFC 3987), recommended by IDTA, is used to uniquely identify the robot and all other assets, such as the aircraft surrogate, in the CPSS. Here it consists of an imaginary DLR domain and a UUID:

Robot ID: **https://dlr.de/mo/asset/e95ea183a87f48d3811c2cae77388044**

Figure 20 shows the corresponding AASX file with the metadata submodels at the top. However, a further description of the metadata (nameplate, technical documentation etc.) used to build the digital representation is beyond the scope of this paper. Here we focus on the proactive part of the AAS: To implement the asset properties in the AAS, the associated SMCs in the middle of the figure link them to the data server and its OPC UA variables via their endpoints, e.g. as shown for the tool center point rotation TCP RX. The skills are collected in the SMC *OperationalData* and the submodels of the tender documents are at the bottom of the figure.

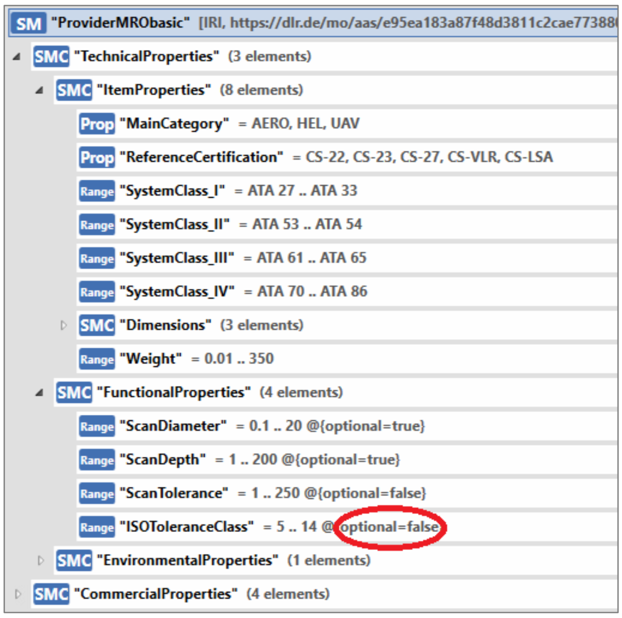


```
AAS "DLR_MO_ROBOT_1" [IRI, https://dlr.de/mo/aas/e95ea183a87f48d3811c2cae77388044] o
  SM "Identification" [IRI, https://dlr.de/mo/aas/e95ea183a87f48d3811c2cae77388044/ids/
  SM "Nameplate" [IRI, https://dlr.de/mo/aas/e95ea183a87f48d3811c2cae77388044/ids/sm
  SM "Documentation" [IRI, https://dlr.de/mo/aas/e95ea183a87f48d3811c2cae77388044/id
  SM "AssetInterfaceDescription" [IRI, https://dlr.de/mo/aas/e95ea183a87f48d3811c2cae77
    SMC "@context" (0 elements) @{@context=https://www.w3.org/2019/wot/td/v1}
    SMC "security" (0 elements) @{security1=nosec_sc}
    SMC "securityDefinitions" (1 elements)
    SMC "properties" (76 elements)
      SMC "monitor_cond_system_flag" (1 elements) @{type=integer} @{readOnly=False}
      SMC "monitor_cond_system_code" (1 elements) @{type=integer} @{readOnly=False}
      SMC "tcp_pose_rz" (1 elements) @{type=integer} @{readOnly=False} @{observable=
      SMC "tcp_pose_rx_degree" (1 elements) @{type=integer} @{readOnly=False} @{obse
        SMC "forms" (1 elements)
          SMC "Form0" (1 elements) @{href=opc.tcp://192.168.20.31:4840/ns=2;i=15}
      SMC "parameter_3" (1 elements) @{type=integer} @{readOnly=False} @{observable
      SMC "parameter_4" (1 elements) @{type=integer} @{readOnly=False} @{observable
      SMC "joint_velocity_wrist3" (1 elements) @{type=integer} @{readOnly=False} @{obs
  SM "AssetProperties" [IRI, https://dlr.de/mo/aas/e95ea183a87f48d3811c2cae77388044/i
  SM "TechnicalData" [IRI, https://dlr.de/mo/aas/e95ea183a87f48d3811c2cae77388044/ids
  SM "OperationalData" [IRI, https://dlr.de/mo/aas/e95ea183a87f48d3811c2cae77388044/i
  SM "ProvideMROadvanced" [IRI, https://dlr.de/mo/aas/e95ea183a87f48d3811c2cae7738
  SM "ProviderMRObasic" [IRI, https://dlr.de/mo/aas/e95ea183a87f48d3811c2cae7738804
  SM "RequestEngineOneTest" [IRI, https://dlr.de/mo/aas/e95ea183a87f48d3811c2cae7738
```

Figure 20: Reduced AASX definition for the UR10e in the proAAS with example of asset interface configuration

For the present use-case, the relevant tender document submodel is marked in Figure 20 below, namely *ProviderMRObasic*. It is the complementary counterpart of the submodel *RequesterMRObasic* send by the aircraft surrogate (see chapter 3.1.1 → Figure 18) and received by the robot. Its more detailed description is shown in Figure 21: the obvious difference between the two is that the service provider offers a range of its capabilities for each property requested. After receiving the aircraft's tender document, the robot's AAS checks that the number, name and type of the properties requested match the expected scheme (bidding state machine → *CapabilityCheck*). If positive, the AAS then checks whether the requested values are within the feasible range:

For example, the MRO provider offers its services in the main categories of Aircraft, Helicopters and Unmanned Aerial Vehicles, which are represented by the *MainCategory* property as a list with AERO, HEL and UAV. The *ReferenceCertification* property is a list of all the certification specifications the MRO provider is allowed to work on, such as CS-VLR for Very Light Rotorcraft. The possible services cover different systems. These are indicated by ATA chapters in the *SystemClass* range properties. For example, SystemClass_I, which corresponds here to the ATA cluster "Aircraft Systems", indicates that the MRO provider only supports work on ATA Chapter 27 ("Flight Controls") to ATA Chapter 33 ("Lights"). In SystemClass_II, "Aircraft Structure", only works on "Fuselage" and "Nacelles" are supported. SystemClass_III and IV comprise elements of the propulsion system. In addition, the MRO provider has limitations on the dimensions and weight of the aircraft and would also refuse the request if the aircraft is outside these limits (e.g. maximum allowable landing weight on platform).



```
SM "ProviderMRObasic" [IRI, https://dlr.de/mo/aas/e95ea183a87f48d3811c2cae77388
  SMC "TechnicalProperties" (3 elements)
    SMC "ItemProperties" (8 elements)
      Prop "MainCategory" = AERO, HEL, UAV
      Prop "ReferenceCertification" = CS-22, CS-23, CS-27, CS-VLR, CS-LSA
      Range "SystemClass_I" = ATA 27 .. ATA 33
      Range "SystemClass_II" = ATA 53 .. ATA 54
      Range "SystemClass_III" = ATA 61 .. ATA 65
      Range "SystemClass_IV" = ATA 70 .. ATA 86
      SMC "Dimensions" (3 elements)
      Range "Weight" = 0.01 .. 350
    SMC "FunctionalProperties" (4 elements)
      Range "ScanDiameter" = 0.1 .. 20 @{optional=true}
      Range "ScanDepth" = 1 .. 200 @{optional=true}
      Range "ScanTolerance" = 1 .. 250 @{optional=false}
      Range "ISOToleranceClass" = 5 .. 14 @{optional=false}
    SMC "EnvironmentalProperties" (1 elements)
  SMC "CommercialProperties" (4 elements)
```

Figure 21: Tender document submodel for a basic MRO with robot as service provider (example: inspection scan)

The tender document also includes functional requirements for the requested service, such as granularity or acceptable tolerances. The MRO service provider also checks whether the service requester's requirements are feasible. However, it must be ensured whether these requirements are mandatory or optional values, as the MRO service provider decides on the selection and settings of the appropriate MRO system in relation to the overall service requested. Both approaches are possible by considering an additional quantifier as a Boolean indicating whether the required functional property is mandatory or not (see marking in Figure 21): The services requester requests an inspection with custom settings where only the tolerance requirements are mandatory. In the use case, all the technical requirements of the MRO requester can be met by the MRO service provider. Before returning the proposal to the MRO requester, the MRO provider balances the commercial properties in terms of expected costs, times (slots and duration) and the optimum location or provisioning. To prove the concept, these are fixed values that will be replaced by dynamic cost-benefit calculations in later development. In the use case and for simplicity, the aircraft surrogate as the service requester accepts the conditions and sends a service order.

The robot's proactive AAS invokes the ordered service by instructing the PLC to load and execute the appropriate program, e.g. "*drone_primary_inspection.urp*". The program is one out of many on the PLC and is linked to its service definition in the AAS. It simply takes pictures for inspection purposes at all engine's positions (Figure 22 left). At a certain point in our scenario, the MRO robot system needs human assistance and the proactive AAS sends a corresponding CfP to the CPSS. In our use case, an HMI system was extended to an Industry 4.0 component by equipping it with its own proactive AAS (see chapter 3.1.3). It responds to the robot's request and directs the human to perform the desired task: safely replace the aircraft surrogate to complete the inspection. When the entire service is completed, the SP sends a completion message to the SR and, if required, a digital certificate of airworthiness such as EASA Form 1

3.1.3. HMI as MRO service supporter

Human involvement in the cyber-physical social system significantly expands the service portfolio. This requires appropriate interfaces that can communicate in the I4.0 language. First, a web-based communication system is under developed. Its idea is based on a chat-like environment where CPSS participants can exchange requests and responses via human-readable messages. The web server was developed in Python using the Flask libraries, with real-time communication between the GUI (browser) and the server via a socket data exchange (Flask-SocketIO). The GUI has only four buttons (Confirm, Reject, Not Understood, Request) and a message box to keep the application simple for the users. An augmented reality device is used to display the digital messages to the user while performing the task in the real world. In the message field the participants address their quotes with an "@". Before one can join the service specific chat room, it is assumed that the bidding process of the requested service has been successfully finished. The selected service provider(s), in our example "MRO HUMAN", will be invited to enter the room and will be given further advice on how to complete the task

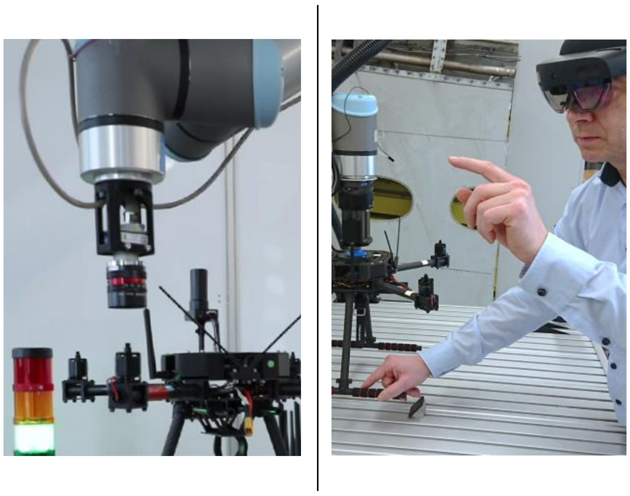
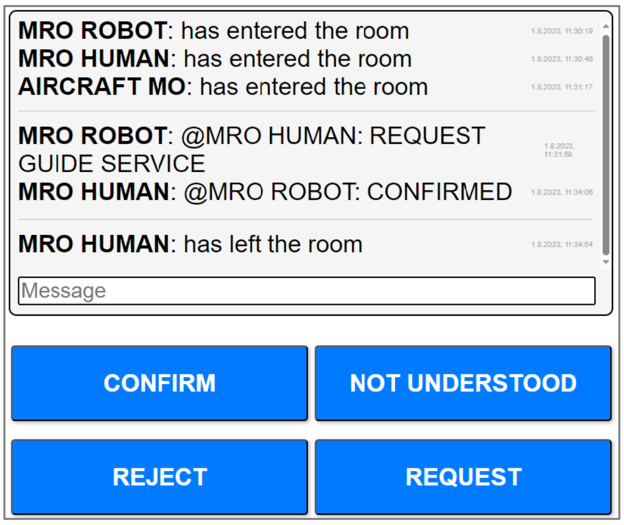


Figure 22: MRO SP inspects MRO SR (left); MRO SP (here robot) requests support by human SP (right)

In fact, this kind of HMI system is treated as an asset and has also been encapsulated with a proactive AAS to ensure its I4.0 compliant integration into the CPSS. In preparation, the HMI has been extended by an OPC UA server, which is connected to the web server. Thus, OPC UA variables can be displayed directly in the web-based GUI and vice versa GUI variables are transmitted to the corresponding OPC UA variables. The OPC UA variables have been implemented in the AAS by specifying their properties and references in


MRO ROBOT: has entered the room 1.8.2023, 11:30:19
MRO HUMAN: has entered the room 1.8.2023, 11:30:46
AIRCRAFT MO: has entered the room 1.8.2023, 11:31:17

MRO ROBOT: @MRO HUMAN: REQUEST GUIDE SERVICE 1.8.2023, 11:31:59
MRO HUMAN: @MRO ROBOT: CONFIRMED 1.8.2023, 11:34:06

MRO HUMAN: has left the room 1.8.2023, 11:34:54

Figure 23: Developed GUI of the proposed HMI system as a chat room like interface with I4.0 compatibility

the AASX file as introduced in chapter 2.1.1. As with the other I4.0 components introduced before, the integrated bidding state machine negotiates the services based on a tender document. It is not yet fully defined, but as shown in Figure 24 the SMC *FunctionalProperties* refer to the capabilities of the human with, where available, additional tools: For example, functionalities such as drilling diameter or depth, when using a hand drill, manual force ranges for lifting or moving, as well as screwing specifications (type, torque) could be defined. Furthermore, only decisions on the continuation of the process could be requested.

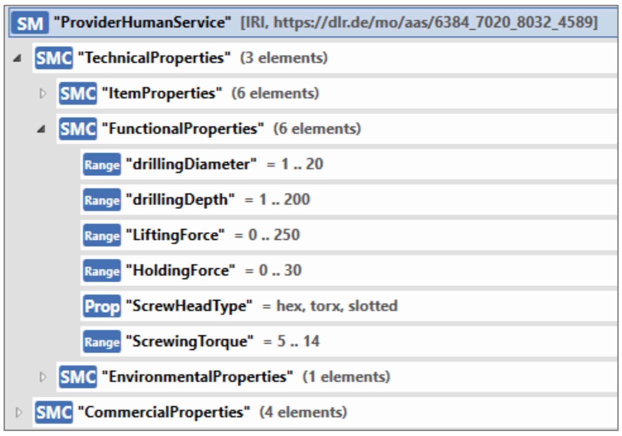

SM "ProviderHumanService" [IRI, https://dlr.de/mo/aas/6384_7020_8032_4589]
- SMC "TechnicalProperties" (3 elements)
  - SMC "ItemProperties" (6 elements)
  - SMC "FunctionalProperties" (6 elements)
    - Range "drillingDiameter" = 1 .. 20
    - Range "drillingDepth" = 1 .. 200
    - Range "LiftingForce" = 0 .. 250
    - Range "HoldingForce" = 0 .. 30
    - Prop "ScrewHeadType" = hex, torx, slotted
    - Range "ScrewingTorque" = 5 .. 14
  - SMC "EnvironmentalProperties" (1 elements)
- SMC "CommercialProperties" (4 elements)

Figure 24: Submodel of the tender document for the request for human assistance by the MRO robot as requester and a human as service provider (example)

After the positive feasibility check of the requirements, the return of the service proposal to the requesting MRO system (robot) and its acceptance, the interaction via the chat room begins. As shown in Figure 24, the MRO robot requests the human for assistance in guiding the aircraft surrogate to its final position (Figure 22 right). After the human acknowledges the request, the task is completed cooperatively. It depends on the design of the MRO system and the I4.0 components whether the human needs to confirm that the task has been completed, or whether the MRO robot decides this automatically. The latter is realised in our use case, and thus the MRO robot continues its inspection process after the repositioning of the aircraft, together with the support of the human, until the end.

4. CONCLUSION AND OUTLOOK

The research questions were whether and how data-driven MRO environments could benefit from Industry 4.0 knowledge. To answer these questions, in addition to a theoretical view, an experimental setup of a CPSS was designed to test MRO-adapted Industry 4.0 technologies. The main finding is that building data-driven MRO environments based on Industry 4.0 knowledge is strong promising. With the Asset Administration Shell (AAS), the I4.0 community has introduced a concept for the realization of standardized "digital twins": The AAS provides both a meta-model for the digital representation of tangible and intangible assets over its life-cycle and the complementary framework for seamless, autonomous and proactive interaction between all participants in a CPSS – cross-stakeholder, cross-company and cross-technology.

In this paper, a Python-based proactive AAS is used to develop Industry 4.0 components (I4.0C) for an exemplary data-driven MRO environment (requester/provider). In line with this, a hardware and software solution are presented to easily equip assets with the minimum technical requirements to operate as I4.0C. This was applied in a use case to an aircraft system surrogate, an MRO service provider and an HMI system for human MRO support. The self-managed interactions in terms of service requests and provisioning are based on a bidding protocol and logics of decision-making. In this protocol, tender documents are exchanged and negotiated between the AAS, which include technical or commercial properties. They need MRO aligned specifications, the first ideas of which are outlined in the paper. In an experimental setup, the successful interoperation of assets in a data-driven MRO environment has been verified. Hereby the introduced proactive ASS BOX was used to create MRO I4.0 components of an aircraft surrogate, an MRO robot and a human-controlled support system. In the use case, the participants solve an inspection task collaboratively with digital exchange of all necessary information and requests via the CPSS network infrastructure. In summary, all steps together show how AAS could be applied to data-driven MRO in aviation.

Further developments will mainly focus on following topics:

a. Implementation of the proactive AAS to develop further MRO assets into I4.0 components.
b. Improving the presented I4.0 components to fully digital representations (static descriptions along with point e.)
c. Development of cost-benefit and decision-support services either as AAS in-build algorithms or independently working solutions as AAS in the CPSS.
d. Development of tender documents (AAS submodel) for the I4.0 based bidding procedure to trade comprehensive MRO services in terms of functional and commercial properties.
e. Integration of digital representations of certificates, manuals, drawings, etc., of the assets based on the adapted IDTA submodel Handover Documentation [20].

5. ABBREVIATIONS

AAS	Asset Administration Shell
CC	Control Component
CfP	Call for Proposal
CPS	Cyber-Physical System
CPSS	Cyber-Physical-Social System
CSS	Capability-Skill-Service
EASA	European Union Aviation Safety Agency
FAA	Federal Aviation Administration
IDTA	Industrial Digital Twin Association e.V.
IIC	Industrial Internet Consortium
IIoT	Industrial Internet of Things
IIRA	Industrial Internet Reference Architecture
IoT	Internet of Things
IT	Information Technology
MaSiMO	Maintenance Simulation Model at DLR-MO
MCDM	Multi-Criteria-Decision-Making
MRO	Maintenance, Repair and Overhaul
OPC UA	Open Platform Communication UA
PI4.0	Plattform Industrie 4.0
Prop	Property
RAMI 4.0	Reference Architectural Model Industry 4.0
Ref	Reference
SCADA	Supervisory Control and Data Acquisition
SM	Submodel
SMC	Submodel Collection
SP	Service Provider
SR	Service Requester
TOPSIS	Technique for Order Preference by Similarity to Ideal Solution

6. REFERENCES

[1] AAS Reference Modelling, **2021**, Discussion Paper, Plattform Industrie 4.0, Berlin, Germany, available online: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/AAS_Reference_Modelling (accessed on 04.08.2023)

[2] Belyaev, A.; Diedrich, C. Aktive Verwaltungsschale von I4.0-Komponenten. **2019**, AUTOMATION – Leitkongress der Mess- und Automatisierungstechnik. VDI Verein Deutscher Ingenieure, Baden-Baden

[3] Belyaev, A.; Diedrich, C. Erhöhung der Flexibilität und Robustheit zwischen Interaktionspartnern durch das Merkmalmodell, **2018**, Automatisierungstechnik, Volume 67, no. 3, pp. 193−207 DOI: 10.1515/auto-2018-0103

[4] Belyaev, A.; Diedrich, C.; Köther, H.; Dogan, A. Dezentraler IOTA-basierter Industrie-Marktplatz: Industrie-Marktplatz auf Basis von IOTA, eCl@ss und I4.0-Verwaltungsschale, **2020**, Industrie 4.0 Management, 36, ISSN: 2364-9208

[5] Communication: 2030 Digital Compass: the European way for the Digital Decade, European Commission, COMMUNICATION FROM THE COMMISSION TO THE EUROPEAN

PARLIAMENT, Brussels, **2021** Available online: https://commission.europa.eu/system/files/2023-01/cellar_12e835e2-81af-11eb-9ac9-01aa75ed71a1.0001.02_DOC_1.pdf (accessed on 18.03.2023)

[6] Cross-industry semantic interoperability, part one. **2017**, Blog, Embedded Computing Design (online print), publisher OpenSystems media, Scottsdale, USA Available online: https://embeddedcomputing.com/technology/iot/cross-industry-semantic-interoperability-part-one (accessed on 20.08.2022)

[7] Details of the Asset Administration Shell Part 2 – Interoperability at Runtime – Exchanging Information via Application Programming Interfaces, Specification document, **2021**, Plattform Industrie 4.0, Publisher: Federal Ministry for Economic Affairs and Energy (BMWi), Berlin, Germany Available online: https://www.plattform-i40.de/IP/Redaktion/DE/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part_2_V1.html (accessed on 19.07.2022)

[8] Diedrich, C. et al. Information Model for Capabilities, Skills & Services, **2021**, Technical Report, WG Semantic and Interaction of I4.0 Components, Plattform Industrie 4.0 DOI: 10.13140/RG.2.2.30098.53440

[9] Diedrich, C.; Belyaev, A. Information Model for Capabilities, Skills & Services, 2022, Kommunikation und Bildverarbeitung in der Automation, Technologien für die intelligente Automation 14, Springer Vieweg, Germany DOI: 10.1007/978-3-662-64283-2_8

[10] Diedrich, C.; Schröder, T.; Belyaev, A. Interoperability of Cyber Physical Systems, 2022, Kommunikation und Bildverarbeitung in der Automation, Technologien für die intelligente Automation 14, Springer Vieweg, Germany DOI: 10.1007/978-3-662-64283-2_8

[11] Digital Twin and Asset Administration Shell Concepts and Application in the Industrial Internet and Industrie 4.0. An Industrial Internet Consortium and Plattform Industrie 4.0 Joint Whitepaper, **2020** Available online: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/Digital-Twin-and-Asset-Administration-Shell-Concepts.html (accessed on 18.03.2023)

[12] DIN EN IEC 63278-1 VDE 0810-781:2022-07 Asset Administration Shell for industrial applications, **2022**, VDE Standards, VDE Verlag, Berlin

[13] DIN SPEC 91345:2016-04 Reference Architecture Model Industrie 4.0 (RAMI 4.0), DIN Deutsches Institut für Normung e. V., **2016**, DOI: 10.31030/2436156

[14] European Research Cluster on the Internet of Things, **2020,** Available online: http://www.internet-of-things-research.eu/index.html (accessed on 18.03.2023).

[15] FAA Joint Aircraft System/Component Code – Table and Definitions, 2008, Federal Aviation Administration, Regulatory Support Division, Oklahoma, USA, available online: https://av-info.faa.gov/sdrx/documents/JASC_Code.pdf (accessed 03.08.2023)

[16] Functional View of the Asset Administration Shell in an Industrie 4.0 System Environment, **2021**, Discussion Paper, Plattform Industrie 4.0, Berlin, Germany, available online: https://www.plattform-i40.de/IP/Redaktion/DE/Downloads/Publikation/Functional-View (accessed on 02.08.2023)

[17] Grunau, S.; Redeker, M.; Göllner, D.; Wisniewski, L. The Implementation of Proactive Asset Administration Shells: Evaluation of Possibilities and Realization in an Order Driven Production. **2022**, In: Jasperneite, J., Lohweg, V. (eds) Kommunikation und Bildverarbeitung in der Automation. Technologien für die intelligente Automation, vol 14. Springer Vieweg, Berlin, Heidelberg DOI: 10.1007/978-3-662-64283-2_10

[18] IDTA 01001-3-0, Specification of the Asset Administration Shell Part 1: Metamodel, Specification, **2023**, publisher: Industrial Digital Twin Association, Frankfurt am Main, Germany, available online: https://industrialdigitaltwin.org (accessed on 02.08.2023)

[19] IDTA 01005-3-0, Specification of the Asset Administration Shell Part 5: Package File Format (AASX), Specification, **2023**, publisher: Industrial Digital Twin Association, Frankfurt am Main, Germany, available online: https://industrialdigitaltwin.org (accessed on 02.08.2023)

[20] IDTA 02004-1-2, Handover Documentation, Submodel Template of the Asset Administration Shell, **2023**, publisher: Industrial Digital Twin Association, Frankfurt am Main, Germany, available online: https://industrialdigitaltwin.org (accessed on 04.08.2023)

[21] IDTA 02015-1-0, Control Component Type, Submodel Template of the Asset Administration Shell, **2023**, publisher: Industrial Digital Twin Association, Frankfurt am Main, Germany, available online: https://industrialdigitaltwin.org (accessed on 02.08.2023)

[22] IDTA 02016-1-0, Control Component Instance, Submodel Template of the Asset Administration Shell, **2023**, publisher: Industrial Digital Twin Association, Frankfurt am Main, Germany, available online: https://industrialdigitaltwin.org (accessed on 02.08.2023)

[23] Jacoby, M.; Jovicic, B.; Stojanovic, L.; Stojanovi´c, N. An Approach for Realizing Hybrid Digital Twins Using Asset Administration Shells and Apache StreamPipes, Article, Information, **2021**, 12, 217. DOI: 10.3390/info12060217

[24] Jacoby, M.; Volz, F.; Weißenbacher, C.; Stojanovic, L.; Usländer, T. An approach for Industrie 4.0-compliant and data-sovereign Digital Twins: Realization of the Industrie 4.0 Asset Administration Shell with a data-sovereignty extension, Article, Automatisierungstechnik, vol. 69, no. 12, **2021**, pp. 1051-1061, DOI: 10.1515/auto-2021-0074

[25] Kagermann, H.; Wahlster, W. Ten Years of Industrie 4.0, **2022**, Sci 2022, 4(3), 26. DOI: 10.3390/sci4030026

[26] Kamtsiuris, A.; Raddatz, F.; Wende, G A Health Index Framework for Condition Monitoring and Health Prediction, **2022**, Conference: 7th European Conference of the PHM Society, Volume 7 DOI: 10.36001/phme.2022.v7i1.3324

[27] Köcher, A.; Belyaev, A.; Hermann, J.; Bock, J.; Meixner, K.; Volkmann, M.; Winter, M.; Zimmermann, P.; Grimm, S.; Diedrich, C. A reference model for common understanding of capabilities and skills in manufacturing, **2023**, Automatisierungstechnik, Volume 71, no. 2, pp. 94-104 DOI: 10.1515/auto-2022-0117

[28] Köcher, A.; Hildebrandt, C.; Vieira da Silva, L.; Fay, A. A Formal Capability and Skill Model for Use in Plug and Produce Scenarios, **2020**, IEEE conference, ETFA, Wien, Austria DOI: 10.1109/ETFA46521.2020.9211874

[29] N.N. Specification Testbed „AAS networked “: Proactive AAS - interaction according to the VDI/VDE 2193, **2019**, Technical Report, University Magdeburg, Institute for Automation Engineering

https://www.lia.ovgu.de/lia_media/LIA_Medien/AASnetworked.pdf (accessed on 27.07.2023)

[30] Pakala, H. K.; Käbisch, S. Integration of asset administration shell and Web of Things, **2021**, DOI: http://dx.doi.org/10.25673/39570 (accessed on 11.07.2023)

[31] SCI4.0, DIN/DKE, German Standardization Roadmap Industrie 4.0, **2020**, Version 4, Berlin: DIN and Frankfurt(M): DKE.
Available online: https://www.din.de/de/forschung-und-innovation/themen/industrie4-0/roadmap-industrie40-62178 (accessed on 18.03.2023)

[32] Structure of the Administration Shell, continuation of the development of the reference model for the Industrie 4.0 component, **2016**, Plattform Industrie 4.0, Working Paper
Available online: https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/structure-of-the-administration-shell.html (accessed on 20.08.2022)

[33] Urban, C.; Belyaev, A.; Diedrich, C. Verwaltungsschale-basierter Ansatz für die Umsetzung von auftragsgesteuerter Produktion, **2022**, Conference paper, 23. VDI-Kongress AUTOMATION – Leitkongress der Mess- und Automatisierungstechnik, Baden-Baden, Germany

[34] VDI/VDE 2193 Part 1:2020-04 Language for I4.0 Components - Structure of messages, VDI/VDE guideline **2020**, VDI-VDE Manual, Düsseldorf

[35] VDI/VDE 2193 Part 2:2020-01 Language for I4.0 components - Interaction protocol for bidding procedures, VDI/VDE guideline **2020**, VDI-VDE Manual, Düsseldorf

[36] Verwaltungsschale in der Praxis, Discussion document, **2020**, Plattform Industrie 4.0, Publisher: Federal Ministry for Economic Affairs and Energy (BMWi), Berlin, Germany, available online: https://www.plattform-i40.de/IP/Redaktion/DE/Downloads/Publikation/2020-verwaltungsschale-in-der-praxis.html (accessed on 19.07.2022)

[37] Vieira da Silva, L. M.; Köcher, A.; Gill, M.; Weiss, M.; Fay, A. Toward a Mapping of Capability and Skill Models using Asset Administration Shells and Ontologies, **2023**, preprint, arXiv, Computer Science
DOI: 10.48550/arXiv.2307.00827

[38] Weiss, M.; Wicke, K.; Wende, G. MaSiMO - A Hybrid Experimental Platform for the Simulation and Evaluation of Data-Driven Maintenance Enterprises, **2022**, Paper, Deutsche Gesellschaft für Luft- und Raumfahrt - Lilienthal-Oberth e.V.
DOI: 10.25967/570154

[39] Winkler, D..; Gill, M. S.; Fay, A. The Asset Administration Shell as a solution concept for the realisation of interoperable Digital Twins of Aircraft Components in Maintenance, Repair and Overhaul, **2022**, Paper, Deutsche Gesellschaft für Luft- und Raumfahrt - Lilienthal-Oberth e.V.
DOI: 10.25967/570274