Implementing a Metadata Manager for Machine Learning with the Asset Administration Shell

1st Alexandre Sawczuk da Silva
Fraunhofer Institute for Cognitive Systems IKS
Munich, Germany
alexandre.sawczuk.da.silva@iks.fraunhofer.de

2nd Hoai My Van
Fraunhofer Institute for Cognitive Systems IKS
Munich, Germany
hoai.my.van@iks.fraunhofer.de

3rd Gereon Weiss
Fraunhofer Institute for Cognitive Systems IKS
Munich, Germany
gereon.weiss@iks.fraunhofer.de

***Abstract*—With the rise of Industry 4.0, businesses are increasingly turning to Machine Learning to leverage data for improving quality and productivity. However, one open challenge when embracing Machine Learning in this context is the integration of cloud infrastructures, as well as the heterogeneity of data, interfaces, and protocols in the production environment. To address this, we are developing a framework that aims to simplify the adoption of Machine Learning techniques for heterogeneous industrial automation systems. One of the core features of this framework is the ability to handle data about production devices – a scenario that is naturally suited to the use of Asset Administration Shells. However, the implementation of a system that uses Asset Administration Shells comes with its own set of challenges, such as the abstraction of details from users and the representation of device topologies. Thus, this paper introduces the concepts and implementation of a Metadata Manager component in the aforementioned framework that uses Asset Administration Shells as its basis. We further examine the Metadata Manager's current structure with unit testing, derive planned extensions, and discuss future directions from the Industry 4.0 perspective.**

***Index Terms*—Asset Administration Shell, BaSyx, AASX Package Explorer, production plant metadata, digital twin**

I. Introduction

The concept of Industry 4.0, which refers to the evolution of manufacturing processes towards a highly automated and adaptable infrastructure [1], has become increasingly popular in recent years. As businesses strive to remain competitive and to adjust to market demands [2], they are turning towards solutions that improve operation quality and productivity. These are objectives that are at the core of Industry 4.0, with its focus on digitalization and on leveraging the resulting data [1]. Naturally, this approach paves the way for the use of emerging Machine Learning (ML) techniques, which is indeed an active area of investigation [3], [4]. However, a number of challenges have been reported when applying these techniques in the context of Industry 4.0, including the need to consider cloud architectures and to handle large amounts of heterogeneous data, diverse system interfaces, and communication protocols [3].

To address these challenges and to provide a solid foundation for the application and life cycle management of ML in industrial automation environments, we propose an adaptive framework to simplify the use of ML techniques, called ACD*C (Automated Continuous Dynamic {Acquisition, Processing, Analysis} Cycle). To accomplish this, the ACD*C framework encapsulates ML algorithms and processes as services, making it possible to deploy key functionality as required. Additionally, it provides standardized connectors to production line devices, harmonizes output data, and supplies key information to assist Data Scientists.

From this, the framework needs to manage metadata about the production devices, their underlying topology, and the settings required for communicating with them. Thus, the ACD*C framework has been designed to include a *Metadata Manager* as one of its key components. As most of this metadata pertains to production devices, the Manager is developed around the concept of the Asset Administration Shell (AAS), which includes standardized data interfaces for Industry 4.0 components [5], [6]. The AAS was chosen because – as the digital reflection of assets – it cannot only naturally track the desired device information, but also ensure that the framework remains interoperable and leverages eventual manufacturer-provided AAS data in the future [7], [8].

The goal of this paper is to examine the process of implementing the Metadata Manager while building upon the concepts of the Asset Administration Shell. Specifically, we see the main contributions of this paper in the following:

- Proposal of a Metadata Manager and its integration with AAS within the framework, for supporting ML life cycle in Industry 4.0.
- Implementation of the Metadata Manager with AAS in an industrial use case in a lab environment.
- Derived findings and planned extensions employing other AAS characteristics.

This project was funded by the Bavarian Ministry for Economic Affairs, Regional Development and Energy and supported by the Bayern Innovativ GmbH as part of the project "REMORA – Multi-Stage Automated Continuous Delivery for AI-based Software & Services Development in Industry 4.0".

- Identification of future directions to be pursued as the AAS continues to mature as a concept and reaches broader adoption.

The remainder of this paper is organized as follows. Section II contains a more detailed description of the ACD*C framework, as well as a discussion of relevant ideas and technologies pertaining to the AAS. Section III presents design decisions and challenges in the implementation of the Metadata Manager. Section IV analyzes current execution time results obtained from unit testing. Section V explores planned extensions to the Metadata Manager. Section VI considers future directions for the framework and for this domain in general. Section VII concludes the paper.

II. Background

*A. ACD*C Framework for Supporting the Data and Machine Learning Life Cycle*

With the aim to simplify the application of ML techniques in the Industry 4.0, our goal is to develop an interoperable framework – the ACD*C framework – that supports and optimizes the data and ML life cycle. This framework supports the application of ML to a wide variety of scenarios, including fault detection, predictive maintenance, and production optimization. Likewise, it enables the use of a broad range of ML techniques, from neural networks and deep learning to reinforcement learning. These techniques can either be used for separate analysis, or for directly influencing the production process (e.g., to improve machine utilization or throughput). The ACD*C framework addresses the three main ML processes for (1) data acquisition, (2) data processing including model training, and (3) data analysis. These stages are realized by so-called *ML-Services*, which are services that encapsulate particular steps in the Machine Learning process (e.g. training). These services can be deployed in a decentralized fashion to devices in the plant. In addition to supporting this flexible integration of services, which facilitates the automation of ML processes, this framework also provides unified access to data and metadata for both ML-Services and Data Scientists. Figure 1 shows a high-level overview of the four components in the ACD*C framework. The *ML-Interface* component enables uniform data access and assists Data Scientists in the creation of ML-Services by giving access to important contextual data and metadata as well as handling any required data transformations; the *ML-Management* component controls the processes of training and redeploying models as required; the *Metadata Management* component provides information about the assets and topology of the production plant, as well as handling metadata related to the ML process; the *Service Management* component handles the creation, deployment, and coordination of services.

A set of selected machines is employed to illustrate a use case for the ACD*C framework, and this setup is referred to throughout this paper. As shown in Figure 2, this modular production use case consists of a series of modular production system (MPS) stations organized as a minimal production line. Each station independently handles a distinct manufacturing step in the assembly of magazines, caps, and rings into a composite final product. In this installation, stations are not connected by conveyor belts, but instead by a robot that moves the partially assembled products from one station to the next. Once the products are fully assembled, they are inspected by a camera to determine whether they match the given specifications (i.e., whether the magazines, caps, and rings of the correct color were used in the assembly process). The ACD*C framework is applied to this use case, with the goal to enable the creation of an ML-Service that performs image-based error detection using the output produced by the camera. Note that the ML-Service that accomplishes this is deployed to a PC connected to the camera.

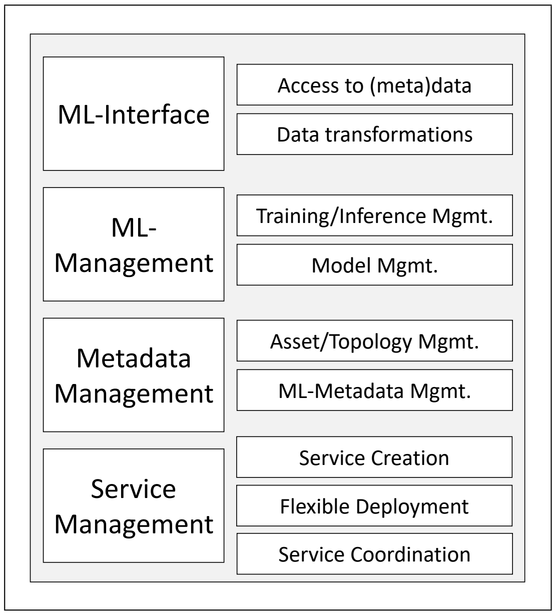


Fig. 1. Overview of components in ACD*C framework.

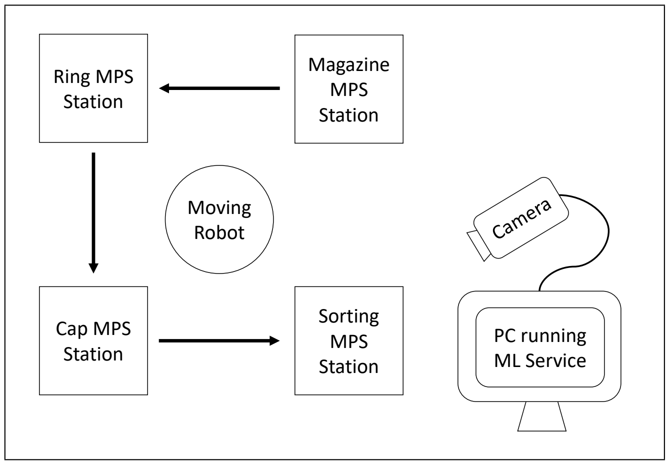


Fig. 2. Diagram with overall representation of the modular production use case.

B. Literature Review

As this project investigates the intersection between the concepts of Industry 4.0, Asset Administration Shell, digital twins, Machine Learning, and related technologies, this literature review focuses on providing a brief explanation and introduction to these various concepts and related work. The motivation for this subsection is to provide an essential set of definitions for the terms used throughout the paper and to discuss how these ideas interrelate.

1) Industry 4.0 and Machine Learning: The term "Industry 4.0" refers to the idea of a fourth industrial revolution [9]. Whereas the first industrial revolution focused on mechanization, the second on the use of electricity, and the third on process digitalization, the vision of Industry 4.0 is to introduce smart objects into the manufacturing process [1]. Specifically, the goal is to further the digitalization of production plants by leveraging the dynamic connection and integration of devices at the shop floor [10]. It is little surprise, then, that Machine Learning is instrumental to the execution of this vision. In fact, research on the use of Machine Learning to the domain of Industry 4.0 is very active, with a particular emphasis on fault detection and predictive maintenance approaches [3]. In order to make use of these techniques, and to support the tenet of Industry 4.0 in general, it is necessary to employ a unified standard to communicate about the capabilities of devices and the data produced by them [11]. With a standardized format, data can be processed in a machine-interpretable manner, leading to the ultimate goal of independent and self-configuring devices.

2) Asset Administration Shell and Digital Twins: To address the need for standardized data about the components of a manufacturing plant, researchers and practitioners developed the concept of the Asset Administration Shell (AAS), which is a software representation of a physical asset in the production line [12]. An AAS is comprised of a number of submodels, each capturing a different information perspective and its corresponding data properties for the underlying asset [13]. Employing the AAS ensures that a consistent information model is used, which in turn supports the interoperability of components [14]. The concept of the AAS is similar to that of digital twins in that both provide detailed digital models of a physical system. However, the AAS can be thought of as an enriched and mature version of a digital twin, where all information and interfaces to the corresponding device are fully available [15]. This availability is crucial for supporting the principles of Industry 4.0 and ML. The work in [16] indicates that the structure and information provided by the AAS support the employment of ML techniques.

3) Supporting Technologies: Supporting tools and technologies have been developed to promote the use of Industry 4.0 and Asset Administration Shell concepts. One of them is the AASX Package Explorer, which is a tool for creating, viewing, and editing .AASX files [17], [18]. The Package Explorer allows for the modelling of assets and the creation of their AAS, as well as the assignment and organization of data properties. It is an intuitive tool for perusing the structure of Asset Administration Shells. Another key technology is BaSyx [19], [20], which is a platform that provides software to assist with the implementation of Industry 4.0 applications. It includes components to assist with the modelling, communication, and data processing for shop floor devices. BaSyx incorporates the concept of AAS into the platform, allowing their dynamic creation and also loading from .AASX files.

4) Metadata Management: Independently from the concepts of Industry 4.0 and AAS, metadata management of software systems has been an area of focus on its own accord. Metadata management is connected to database technologies, where information such as schema mappings must be efficiently stored and handled [21]. Research in this field is ongoing, with works in the area evolving from predominantly discussing the speed of data access [22] to presenting broader architectural and technological ideas [23]. One of these proposals involves a data lake, which is a platform for handling heterogeneous data sets in a unified manner. The architecture of a data lake introduces ideas such as data ingestion, cleaning, and versioning [24], all of which are relevant when implementing a metadata management solution. Still, the focus of metadata management in the context of industrial automation systems is not on handling big data (as it is the case with data lakes), but on supporting the digitalization and automation of industrial processes by providing crucial information. Namely, the work in this area seeks to standardize data so that it can be freely exchanged among enterprise systems [25].

5) Service-Oriented Architecture: Another fundamental concept that must be mentioned is that of Service-Oriented Architectures (SOA), which is consistently used as a fundamental paradigm by our ACD*C framework and ultimately connects to the overall notion of Industry 4.0. SOA organizes components as services, resulting in a flexible system that can easily be updated or reconfigured as necessary [26], [27]. This architecture echoes Industry 4.0's objective of reaching a modular and interoperable ecosystem for smart manufacturing [28], constituting an area of active research. As example, the work in [29] clearly connects the two ideas, pointing out that robots, machines, and applications can be made available as services for use in the manufacturing process. These services can be flexibly discovered and composed, and they communicate via a standardized service bus. Likewise, the work in [30] explores the compatibility between service-based cloud computing and Industry 4.0 standards, concluding that manufacturing services can successfully be used via a cloud-based interface. Thus, SOA can be seen as instrumental in the implementation of the infrastructure required for realizing Industry 4.0 concepts.

III. Metadata Manager

The Metadata Manager was implemented in Java, making use of the BaSyx platform [19]. It provides a standardized interface that can be used by the Service Management component (cf. Figure 1) to provide and retrieve metadata about devices in the production plant. This data is used to generate appropriate connectors to communicate with devices, to check resources before deploying ML-Services, and to provide useful information to Data Scientists – as they develop specific ML-Services within the ACD*C framework. The Metadata Manager will also be extended to track metadata related to ML experiments. Building upon BaSyx's structure, the introduced Metadata Manager uses the AAS to structure and store device data. To do so, it instantiates a BaSyx server and registry to manage the created shells. As summarized in Table I, the current interface for the Metadata Manager allows the user to create device representations (either by manually providing the information or by importing it from a .AASX file), remove them, and retrieve information about the data variables (i.e. communication properties) of each device. Additionally, the Metadata Manager interface allows users to set topological connections between devices (once again, either manually or by importing an .AASX file), remove them, and retrieve a graph-based topology representation. The following subsections discuss in more detail the use of AAS in the implementation of the Metadata Manager, addressing three parts of the component. For each subsection, a key concern regarding the use of AAS for the implementation is highlighted. Throughout this section, screenshots obtained with the AASX Package Explorer are used to provide data examples based on the previously presented modular production use case.

TABLE I
OVERVIEW FOR THE METADATA MANAGER'S CURRENT INTERFACE.

Method Name and Parameters	Description and Return Type
`createDevice` • `String deviceName` • `String ipAddress` • `boolean deploymentHost` • `List<DataVariable> dataVariableList`	Creates a new device with the specified information. • Returns `boolean`
`removeDevice` • `String deviceName`	Removes the specified device. • Returns `boolean`
`importDevices` • `String fileName`	Creates devices based on the descriptions contained in a .AASX file. • Returns `boolean`
`checkDataVariableExists` • `String deviceName` • `String dataVariableName`	Checks whether a given data variable has been defined for a device. • Returns `boolean`
`getDataVariableInformation` • `String deviceName` • `String dataVariableName`	Retrieves the details for a given data variable. • Returns `DataVariable`
`getConnectorSettings` • `String protocol`	Provides the connector settings according to the communication protocol. • Returns `ConnectorDetails`
`getIPAddress` • `String deviceName`	Retrieves the IP address for a given device. • Returns `String`
`createTopologyRelationship` • `String firstDevice` • `String secondDevice`	Creates a topology relationship between devices. • Returns `boolean`
`removeTopologyRelationship` • `String firstDevice` • `String secondDevice`	Removes the topology relationship between devices. • Returns `boolean`
`getTopology`	Retrieves a graph representation of the current topology for existing devices. • Returns `TopologyGraph`
`importTopology` • `String fileName`	Creates topology connections based on the descriptions in a .AASX file. • Returns `boolean`

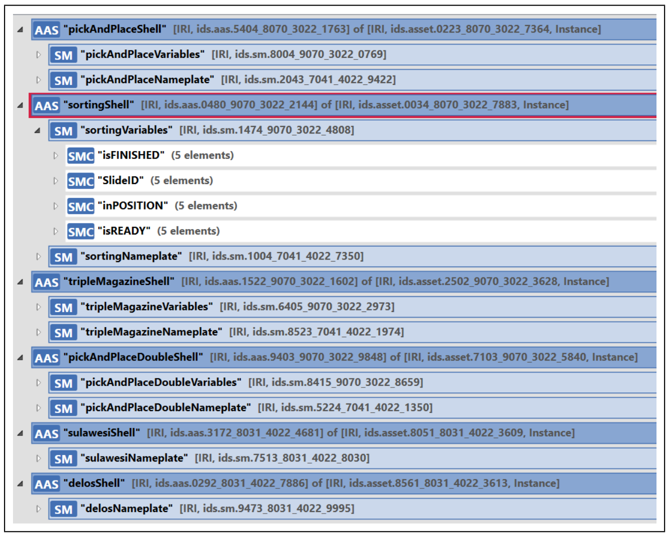
A. Basic Device Information

The methods related to the creation, removal, and import of devices were implemented using the AAS functionality provided by BaSyx. Figure 3 shows the .AASX file, created using the AASX Package Explorer, to describe the Asset Administration Shells for the devices that make up the modular production use case. This example showcases how each AAS is organized for a device. Namely, all shells have one obligatory *nameplate submodel*, which captures basic information about the device (e.g., its name, IP address, voltage, etc), and one optional *variables submodel*, which stores the details of the variables to be used when communicating with the device (including details such as the variable type, associated register address, and communication protocol). The *variables submodel* is optional because certain devices (e.g., PCs) do not require any specific variables for communication. As mentioned before, devices can be created either directly by the user or by importing the contents of a .AASX file.

The creation, retrieval, and removal of this information has a straightforward implementation, as it leverages functionality already provided by BaSyx. The main concern for this part of the implementation was in abstracting away the AAS and submodel details from users, so that it is not necessary for them to understand the details of the BaSyx platform when using the Metadata Manager. At times, this was accomplished by defining and using classes for holding specific categories of information (e.g., a *DataVariable* class).

B. Topology

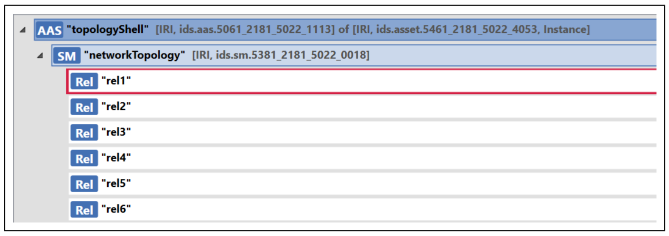
Topological information is useful for the ACD*C framework because it gives further insight as to where to deploy ML-Services and where useful ML data flows through. Additionally, this topology information could be used to optimize how devices are interconnected in the production plant. The methods for defining, importing, and removing topology connections between devices are also implemented using AAS concepts. Specifically, each topological connection is represented as a *RelationshipElement*, which captures an association between two elements in the model. In this context, having this association means that the two devices can communicate with one another. The *RelationshipElement* contains direct references, based on unique identifiers, to the shells for each



```
AAS "pickAndPlaceShell" [IRI, ids.aas.5404_8070_3022_1763] of [IRI, ids.asset.0223_8070_3022_7364, Instance]
  SM "pickAndPlaceVariables" [IRI, ids.sm.8004_9070_3022_0769]
  SM "pickAndPlaceNameplate" [IRI, ids.sm.2043_7041_4022_9422]
AAS "sortingShell" [IRI, ids.aas.0480_9070_3022_2144] of [IRI, ids.asset.0034_8070_3022_7883, Instance]
  SM "sortingVariables" [IRI, ids.sm.1474_9070_3022_4808]
    SMC "isFINISHED" (5 elements)
    SMC "SlideID" (5 elements)
    SMC "inPOSITION" (5 elements)
    SMC "isREADY" (5 elements)
  SM "sortingNameplate" [IRI, ids.sm.1004_7041_4022_7350]
AAS "tripleMagazineShell" [IRI, ids.aas.1522_9070_3022_1602] of [IRI, ids.asset.2502_9070_3022_3628, Instance]
  SM "tripleMagazineVariables" [IRI, ids.sm.6405_9070_3022_2973]
  SM "tripleMagazineNameplate" [IRI, ids.sm.8523_7041_4022_1974]
AAS "pickAndPlaceDoubleShell" [IRI, ids.aas.9403_9070_3022_9848] of [IRI, ids.asset.7103_9070_3022_5840, Instance]
  SM "pickAndPlaceDoubleVariables" [IRI, ids.sm.8415_9070_3022_8659]
  SM "pickAndPlaceDoubleNameplate" [IRI, ids.sm.5224_7041_4022_1350]
AAS "sulawesiShell" [IRI, ids.aas.3172_8031_4022_4681] of [IRI, ids.asset.8051_8031_4022_3609, Instance]
  SM "sulawesiNameplate" [IRI, ids.sm.7513_8031_4022_8030]
AAS "delosShell" [IRI, ids.aas.0292_8031_4022_7886] of [IRI, ids.asset.8561_8031_4022_3613, Instance]
  SM "delosNameplate" [IRI, ids.sm.9473_8031_4022_9995]
```

Fig. 3. Screenshot of the .AASX file that describes devices in the modular production use case.

device. One of the key decisions, when implementing the topology logic, concerned the storage of this data. On the one hand, all topological connections could be tracked in a submodel of an abstract AAS. On the other hand, they could be included as a shared submodel of each device's AAS. Eventually, the decision was made to adopt the former option, as the implementation logic was more intuitive when maintaining the submodel under a specific AAS. Figure 4 shows an example .AASX file, containing the topology information for the modular production use case. While this option is easier to work with, it requires the creation of an abstract "topology" shell, which is a logical component rather than a physical device in the production plant [31].



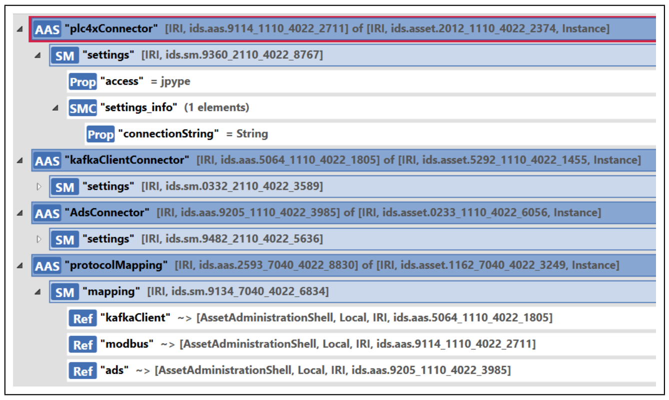
```
AAS "topologyShell" [IRI, ids.aas.5061_2181_5022_1113] of [IRI, ids.asset.5461_2181_5022_4053, Instance]
  SM "networkTopology" [IRI, ids.sm.5381_2181_5022_0018]
    Rel "rel1"
    Rel "rel2"
    Rel "rel3"
    Rel "rel4"
    Rel "rel5"
    Rel "rel6"
```

Fig. 4. Screenshot of the .AASX file with the topology relationships for the modular production use case.

Other concerns in the implementation of the topology have to do with the graph-like nature of this data. One challenge is in handling fully connected topologies, which not only result in an explosion of relationships, but they are also tedious to set up. Our ACD*C framework addresses this by providing users with a shorthand, where a relationship can be created between a device ID and an empty ID. This means that the specific device is fully connected to all other registered devices. Finally, another set of challenges is in ensuring that all recorded relationships remain valid. For instance, if a device is removed, so should all the relationships that include that device. As another example, a relationship between two devices should not be created more than once. To address this, the ACD*C framework uses an internal graph representation that tracks all current edges (i.e., device relationships). Whenever performing a modification that affects the topology state, the code checks against/updates this internal graph to prevent these issues.

C. *Connector Information*

The ACD*C framework also maintains information that is used for creating the connectors that interact with the devices in the production plant. This information is provided to users based on the connectors currently made available in the framework, which are mapped to a specific communication protocol. As shown in Figure 5, this information was also represented using AAS, as connectors fall under the category of logical production components. An AAS was created for each type of connector, containing a settings submodel that holds information on the names and types of the arguments the connector requires, as well as the module used to access the connector code. Additionally, a separate AAS keeps track of the mapping between the communication protocol used and the corresponding connection that should be instantiated (this association is recorded using a *ReferenceElement* that points to the a specific connector's AAS).



```
AAS "plc4xConnector" [IRI, ids.aas.9114_1110_4022_2711] of [IRI, ids.asset.2012_1110_4022_2374, Instance]
  SM "settings" [IRI, ids.sm.9360_2110_4022_8767]
    Prop "access" = jpype
    SMC "settings_info" (1 elements)
      Prop "connectionString" = String
AAS "kafkaClientConnector" [IRI, ids.aas.5064_1110_4022_1805] of [IRI, ids.asset.5292_1110_4022_1455, Instance]
  SM "settings" [IRI, ids.sm.0332_2110_4022_3589]
AAS "AdsConnector" [IRI, ids.aas.9205_1110_4022_3985] of [IRI, ids.asset.0233_1110_4022_6056, Instance]
  SM "settings" [IRI, ids.sm.9482_2110_4022_5636]
AAS "protocolMapping" [IRI, ids.aas.2593_7040_4022_8830] of [IRI, ids.asset.1162_7040_4022_3249, Instance]
  SM "mapping" [IRI, ids.sm.9134_7040_4022_6834]
    Ref "kafkaClient" ~> [AssetAdministrationShell, Local, IRI, ids.aas.5064_1110_4022_1805]
    Ref "modbus" ~> [AssetAdministrationShell, Local, IRI, ids.aas.9114_1110_4022_2711]
    Ref "ads" ~> [AssetAdministrationShell, Local, IRI, ids.aas.9205_1110_4022_3985]
```

Fig. 5. Screenshot of the .AASX file with the connector information for the modular production use case.

For this part of the implementation, the main concern has to do with the current mechanism for the creation of connectors. Namely, they are created by the Service Manager component and not connected back to the Metadata Manager, meaning that AAS cannot directly use the connectors to retrieve the current state of the underlying devices and to communicate with them. This will be addressed in upcoming updates to the implementation of the Metadata Manager.

IV. Unit Testing Results

As part of the implementation process, JUnit tests were created to validate the functionality of the interface provided by the Metadata Manager. While these tests cannot be used as direct evidence regarding the effectiveness of the implementation, they do reveal general efficiency trends that can be used as feedback for further improvement. JUnit 4 tests were run via Eclipse version 2019-12 (4.14.0), on an Ubuntu 20.04.3 LTS virtual machine (VM) with 1024 MB RAM. The VM runs on a Windows 10 Enterprise host with 16 GB RAM, using a single core of the host's quad-core 2.80 GHz processor. Tests were executed 10 independent times, with a warm-up run beforehand.

Table II presents the mean execution time of the "happy path" (i.e., the default usage scenario) for each method offered by the Metadata Manager's interface. Note that the time recorded for each test excludes the overhead associated with instantiating the Metadata Manager, as well as starting/stopping the server and registry used for AAS management. The results show that, in general, the methods for setting and retrieving basic device information (e.g., creating and removing devices, or retrieving data variable information) have roughly similar execution times. In this group, the *getDataVariableInformation* method has the slowest execution time, as it must access multiple submodel values (which incurs some communication latency with the server). Meanwhile, the method for retrieving connector information also performs similarly to those in the previous group. The methods related to the creation and management of topology connections, on the other hand, have noticeably slower execution times. The *importTopology* method, in particular, requires at least twice as long to execute as all other methods, even when compared to the *importDevices* method (which is structurally similar in its implementation). Likewise, the *removeTopologyRelationship* and *getTopology* methods both take longer to execute than methods in other groups.As these results show, rethinking the topology representation is important to improve the performance of the Metadata Manager in upcoming versions.

TABLE II
MEAN EXECUTION TIMES FOR UNIT TESTS (MS), WITH CORRESPONDING STANDARD DEVIATION. VALUES ARE ROUNDED TO ONE DECIMAL PLACE.

Test Name	**Mean $\pm$ Std (ms)**
`testCreateDevice`	414.2 ± 107.6
`testRemoveDevice`	604.8 ± 366.9
`testImportDevices`	593.5 ± 180.4
`testCheckDataVariableExists`	419.1 ± 153.0
`testGetDataVariableInformation`	701.0 ± 199.3
`testGetConnectorSettings`	390.2 ± 93.4
`testGetIPAddress`	475.1 ± 55.5
`testCreateTopologyRelationship`	562.8 ± 127.0
`testRemoveTopologyRelationship`	2075.0 ± 836.1
`testGetTopology`	1637.7 ± 331.9
`testImportTopology`	3467.2 ± 462.3

V. Findings and Planned Extensions

In addition to the need to refine certain aspects of the current implementation, the discussion and assessment of the present Metadata Manager has revealed two interesting extension possibilities to the component, as follows.

A. Dynamic Asset Administration Shell

The first planned extension is to upgrade the AAS from static information containers to data models that are periodically updated when the state of the underlying physical device changes. As discussed in the introductory examples presented in the BaSyx documentation [32], this can be accomplished by assigning lambda functions to the properties that should be dynamically updated. These functions then invoke logic that interfaces with the device (either the actual physical device or a simulated one). In the context of the ACD*C framework, this raises two important questions. The first question, which was already mentioned during discussions about the current implementation, is how to tackle the creation of connectors within the framework. As the BaSyx example shows, the properties in the AAS must have access to the device interface (in the case of the ACD*C framework, the connector). However, the platform currently handles the connector creation in a different module. Thus, the overall framework's architecture must be adapted so that the Metadata Manager either handles the creation of connectors, or at least has direct access to them. The second question is how the ACD*C framework can offer a general-purpose mechanism that allows users to define and customize the state-related information they would like to dynamically track for each device. While that is straightforward in the BaSyx example (as there is a single temperature variable), this requires knowledge of what information is exposed by the different devices in the production plant. Thus, the ACD*C framework must be extended to either import metadata related to device states, or at least allow users to define it.

B. Exposing State Data

Once the first planned extension is carried out, the ACD*C framework should be able to track state updates of all machines. This is valuable data, since it provides insight into the functioning of the production plant, whether any unexpected states have been observed, and what their root cause is. Indeed, this information can be leveraged, with the help of ML, to support tasks such as predictive maintenance, anomaly detection, and production optimization (e.g., ensuring that no machine is underutilized). In order to train ML models that offer these capabilities, the data should be provided in a time series format [3]. Figure 6 shows a visual representation of this concept. In this example, the current state of each device in the modular production use case is tracked in the form of a finite state machine representation. The current states of all devices are grouped by time, thus providing a snapshot of the overall behavior of the production plant at any given moment.

To incorporate these conceptual extensions into the ACD*C framework, the exact representation of a state must be decided upon. The key question is whether the states are automatically inferred based on the current values of selected internal variables, or whether an explicit description is provided for the possible states and transitions. While the first option allows for a simpler implementation, it does not encode any domain-specific knowledge about the behavior of devices. The second

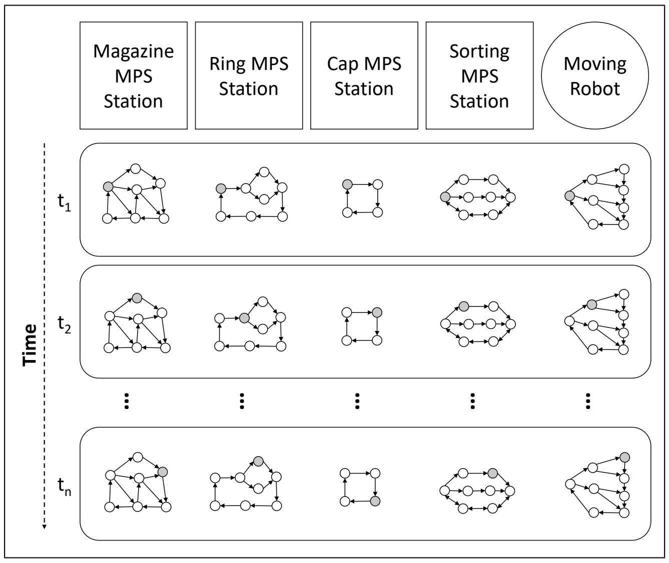


Fig. 6. Conceptual example of state time series for the modular production use case. Note that the finite state machine representations in this example are not accurate depictions of the devices' functions.

option, on the other hand, requires all general device states and transitions between these states to be modelled explicitly. The additional modelling effort in this scenario can be significant, depending on the behavior complexity for the device. However, this involves a direct understanding of its logical functioning, which in turn improves data interpretability.

The approach of explicitly modelling device states has been adopted by the BaSyx platform, with the introduction of control components. These provide a way of controlling physical devices via a service-based interface [33]. Control components are particularly relevant to this discussion because they employ finite state machines to track the status of devices [34]. While some of these state machines only provide general information, such as the error state (i.e., whether the operation is normal or an error has occurred) or the occupation state (i.e., whether the device is currently free or occupied by a different entity), others give specific insight into the device's behavior, such as the execution state (i.e., whether a device is idle, executing a job, suspended, etc.). Additionally, the control component supports the execution of user-created state machines in parallel with the standard ones, which can complement the information already available. Thus, the derived plan for extending the Metadata Manager is to periodically track this state data and make it available for users of the ACD*C framework.

VI. Future Directions

The issues discussed in the previous sections hint at planned extensions for the Metadata Manager and ACD*C framework. However, there are also more general future directions for the framework. Following the discussion of state representations for devices, one interesting prospect is to further expand on this idea by investigating ways of including a description of the functionality of devices in the Asset Administration Shell. Over time, this means that device manufacturers could also include standardized information about the device's specific states in the AAS, which would be useful for testing, simulations, and further simplifying the creation of a digital twin. Indeed, this idea is being actively investigated, with some work focusing on describing state information and then including it into the AAS for later consumption by other tools [35]. Eventually, rather than manually modelling the states of devices, which could become a complex and time consuming endeavor, the finite state machine could be automatically abstracted from the device's data and communication patterns. Once again, this is another active area of investigation, with researchers exploring methods to translate the functionality of programmable logic controllers (PLCs) into state machine representations [36]. In general, these developments steer the work in the field towards the ultimate goal of reaching a fully matured representation of a digital twin, which seamlessly connects static information with dynamic functionality.

Another interesting development is the incorporation of additional testing information to the Asset Administration Shell. For instance, descriptions of test cases could be included, with required input, expected output, and the sequence of states that should be observed as the device operates to fulfill the task. Making this information available would be useful for diagnostics, troubleshooting device configurations, or for ensuring that it behaves as specified. In the case of the ACD*C framework, this information would also be potentially useful as training data for ML-based fault detection. While an exact description of this idea was not found in the literature, the area of verification and validation of industrial systems with the help of digital twins shows similarities [37], [38]. In particular, existing approaches seek to perform this validation via testing or formal verification techniques. This area seems to be under-explored at the moment [37], though further work in this domain could eventually lead to the development of approaches for safety and reliability assessment at runtime.

VII. Conclusion

This paper introduced the implementation of the Metadata Manager, which is a component of the ACD*C framework for supporting the life cycle of Machine Learning in Industry 4.0. The Metadata Manager was built using the concept of Asset Administration Shells and leveraging the existing logic offered by the BaSyx platform. The functionality provided by the Metadata Manager can be conceptualized into three parts: methods that provide and retrieve basic information about devices in the production line, methods for creating, deleting, and exploring topological connections between devices, and logic for retrieving information for the creation of communication connectors for the framework. Executed unit test results show that the efficiency of the current implementation of the Metadata Manager can be improved, especially where the topology-related methods are concerned. Additionally, analyses on the planned extensions to the Metadata Manager indicate that the tracking of dynamic device states and the sharing of this data with the framework's users is a high-value next step. In addition to these planned extensions, future work will include a thorough evaluation of the proposed approach and additional discussions on domain-specific ML applications.

References

[1] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & information systems engineering*, vol. 6, no. 4, pp. 239–242, 2014.

[2] F. E. Garcia-Muiña, R. González-Sánchez, A. M. Ferrari, and D. Settembre-Blundo, "The paradigms of industry 4.0 and circular economy as enabling drivers for the competitiveness of businesses and territories: The case of an italian ceramic tiles manufacturing company," *Social Sciences*, vol. 7, no. 12, p. 255, 2018.

[3] A. Angelopoulos, E. T. Michailidis, N. Nomikos, P. Trakadas, A. Hatziefremidis, S. Voliotis, and T. Zahariadis, "Tackling faults in the industry 4.0 era—a survey of machine-learning solutions and key aspects," *Sensors*, vol. 20, no. 1, p. 109, 2019.

[4] I. S. Candanedo, E. H. Nieves, S. R. González, M. Martín, and A. G. Briones, "Machine learning predictive model for industry 4.0," in *International Conference on Knowledge Management in Organizations*. Springer, 2018, pp. 501–510.

[5] E. Tantik and R. Anderl, "Integrated data model and structure for the asset administration shell in industrie 4.0," *Procedia Cirp*, vol. 60, pp. 86–91, 2017.

[6] P. Marcon, C. Diedrich, F. Zezulka, T. Schröder, A. Belyaev, J. Arm, T. Benesl, Z. Bradac, and I. Vesely, "The asset administration shell of operator in the platform of industry 4.0," in *2018 18th international conference on mechatronics – Mechatronika (ME)*. IEEE, 2018, pp. 1–5.

[7] E. Tantik and R. Anderl, "Potentials of the asset administration shell of industrie 4.0 for service-oriented business models," *Procedia CIRP*, vol. 64, pp. 363–368, 2017.

[8] D. Lang, S. Grunau, L. Wisniewski, and J. Jasperneite, "Utilization of the asset administration shell to support humans during the maintenance process," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, vol. 1. IEEE, 2019, pp. 768–773.

[9] R. Morrar, H. Arman, and S. Mousa, "The fourth industrial revolution (industry 4.0): A social innovation perspective," *Technology Innovation Management Review*, vol. 7, no. 11, pp. 12–20, 2017.

[10] A. Telukdarie and M. N. Sishi, "Enterprise definition for industry 4.0," in *2018 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*. IEEE, 2018, pp. 849–853.

[11] B. Barig, K. Balzereit, and T. Hutschenreuther, "Applying OPC-UA for factory-wide industrial assistance systems," in *2019 15th IEEE International Workshop on Factory Communication Systems (WFCS)*. IEEE, 2019, pp. 1–4.

[12] M. Wenger, A. Zoitl, and T. Müller, "Connecting PLCs with their asset administration shell for automatic device configuration," in *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*. IEEE, 2018, pp. 74–79.

[13] S. R. Bader and M. Maleshkova, "The semantic asset administration shell," in *International Conference on Semantic Systems*. Springer, 2019, pp. 159–174.

[14] D. Stock, M. Schneider, and T. Bauernhansl, "Towards asset administration shell-based resource virtualization in 5G architecture-enabled cyber-physical production systems," *Procedia CIRP*, vol. 104, pp. 945–950, 2021.

[15] C. Wagner, J. Grothoff, U. Epple, R. Drath, S. Malakuti, S. Grüner, M. Hoffmeister, and P. Zimermann, "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, pp. 1–8.

[16] D. Lang, P. Wunderlich, M. Heinz, L. Wisniewski, J. Jasperneite, O. Niggemann, and C. Röcker, "Assistance system to support troubleshooting of complex industrial systems," in *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*. IEEE, 2018, pp. 1–4.

[17] "AASX Package Explorer," https://github.com/admin-shell-io/aasx-package-explorer, accessed on 31 May 2022.

[18] A. Deuter and S. Imort, "PLM/ALM integration with the asset administration shell," *Procedia Manufacturing*, vol. 52, pp. 234–240, 2020.

[19] "Eclipse Foundation: BaSyx," https://wiki.eclipse.org/BaSyx, accessed on 31 May 2022.

[20] K. Esper and F. Schnicke, "Evaluation of the maintainability aspect of industry 4.0 service-oriented production," in *2020 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)*. IEEE, 2020, pp. 8–14.

[21] P. G. Kolaitis, "Schema mappings, data exchange, and metadata management," in *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2005, pp. 61–75.

[22] S. A. Brandt, E. L. Miller, D. D. Long, and L. Xue, "Efficient metadata management in large distributed storage systems," in *20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings*. IEEE, 2003, pp. 290–298.

[23] P. Sawadogo and J. Darmont, "On data lake architectures and metadata management," *Journal of Intelligent Information Systems*, vol. 56, no. 1, pp. 97–120, 2021.

[24] F. Nargesian, E. Zhu, R. J. Miller, K. Q. Pu, and P. C. Arocena, "Data lake management: challenges and opportunities," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 1986–1989, 2019.

[25] R. Eichler, C. Giebler, C. Gröger, E. Hoos, H. Schwarz, and B. Mitschang, "Enterprise-wide metadata management: An industry case on the current state and challenges," in *Business Information Systems*, 2021, pp. 269–279.

[26] K. B. Laskey and K. Laskey, "Service oriented architecture," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 1, no. 1, pp. 101–105, 2009.

[27] D. Sprott and L. Wilkes, "Understanding service-oriented architecture," *The Architecture Journal*, vol. 1, no. 1, pp. 10–17, 2004.

[28] K. Suri, A. Cuccuru, J. Cadavid, S. Gerard, W. Gaaloul, and S. Tata, "Model-based development of modular complex systems for accomplishing system integration for industry 4.0." in *International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 2017, pp. 487–495.

[29] J. Z. Reis and R. F. Gonçalves, "The role of internet of services (IoS) on industry 4.0 through the service oriented architecture (SOA)," in *IFIP International Conference on Advances in Production Management Systems*. Springer, 2018, pp. 20–26.

[30] G. Pedone and I. Mezgár, "Model similarity evidence and interoperability affinity in cloud-ready industry 4.0 technologies," *Computers in industry*, vol. 100, pp. 278–286, 2018.

[31] "BaSyx Documentation: Asset Administration Shell," https://wiki.eclipse.org/BaSyx_/_Documentation_/_AssetAdministrationShell, accessed on 1 June 2022.

[32] "BaSyx: Introductory Examples," https://wiki.eclipse.org/BaSyx_/_Introductory_Examples, accessed on 2 June 2022.

[33] T. Kuhn, F. Schnicke, and P. O. Antonino, "Service-based architectures in production systems: Challenges, solutions & experiences," in *2020 ITU Kaleidoscope: Industry-Driven Digital Transformation (ITU K)*. IEEE, 2020, pp. 1–7.

[34] "BaSyx Documentation: ControlComponent," https://wiki.eclipse.org/BaSyx_/_Documentation_/_API_/_ControlComponent, accessed on 3 June 2022.

[35] D. Göllner, T. Pawlik, and T. Schulte, "Utilization of the asset administration shell for the generation of dynamic simulation models," in *2021 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*. IEEE, 2021, pp. 808–812.

[36] D. Chivilikhin, S. Patil, K. Chukharev, A. Cordonnier, and V. Vyatkin, "Automatic state machine reconstruction from legacy programmable logic controller using data collection and sat solver," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 12, pp. 7821–7831, 2020.

[37] A. Löcklin, M. Müller, T. Jung, N. Jazdi, D. White, and M. Weyrich, "Digital twin for verification and validation of industrial automation systems–a survey," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2020, pp. 851–858.

[38] K. Worden, E. Cross, R. Barthorpe, D. Wagg, and P. Gardner, "On digital twins, mirrors, and virtualizations: Frameworks for model verification and validation," *ASCE-ASME Journal of Risk and Uncertainty in Engineering Systems, Part B: Mechanical Engineering*, vol. 6, no. 3, 2020.